JAVIER GUERRERO JUAN CARLOS GONZALEZ

Descubre cómo ahorrar memoria,
ganar más velocidad de ejecución
componer música a una, dos o tres voces
organizar tus propios programas de gestión,...

LOS SECRETOS DEL



CONTIENE 20 PROGRAMAS

MANHATTAN TRANSFER S.A.

JAVIER GUERRERO · JUAN CARLOS GONZALEZ

LOS SECRETOS DEL



MANHATTAN TRANSFER S.A.

LOS SECRETOS DEL MSX

Primera edición: julio de 1985

Editado por Manhattan Transfer, S.A.
Roca i Batlle, 10-12 – 08023 Barcelona – Tel. 211 22 56
Fotocomposición y Fotomecánica: Ungraf, S.A.
C/. Pujadas, 77-79, 8.°
Impresión: Megra, S.A.
ISBN: 84-86465-00-1
D.L.B. 28875-85

INTRODUCCION

Este libro está escrito especialmente para aquellos que han adoptado el sistema estándar MSX para iniciarse en los secretos de la programación. La elección de esta norma no podía ser más afortunada pues creemos firmemente en la necesidad de un estándar que terminase de una vez con la terrible Babel que rodea tanto a profesionales como aficionados de la informática personal.

El objetivo es que este libro sea un complemento ideal del manual del usuario, de modo que explique con detalle todos aquellos puntos que creemos quedan oscuros y son imprescindibles de conocer para sacar un buen rendimiento al micro ordenador.

En cada uno de los capítulos de este libro veremos con especial detalle todas las instrucciones que son peculiares del sistema MSX y analizaremos en profundidad los dos potentes lenguajes auxiliares de programación, el generador de sonidos y el macrolenguaje de gráficos. También veremos las posibilidades de ampliación del sistema por medio de periféricos.

Finalmente completa «LOS SECRETOS DEL MSX» una gran cantidad de programas relacionados directamente con la teoría expuesta y que serán los que en definitiva nos enseñarán a dominar el potente lenguaje MSX.

J.C.G. y F.J.G.

GENERALIDADES

DEL ORDENADOR PERSONAL AL ORDENADOR DOMESTICO

Supongamos que usted tiene un ordenador personal; supongamos que también tiene un amigo y supongamos que a su vez ese amigo tiene otro ordenador. Todos estos supuestos concurren en el caso de los autores de este libro. Si imaginamos todo esto nos encontraremos con un cuadro realmente cotidiano, pero no por ello menos trágico. Los programas de uno difícilmente correrán en el micro del otro, será imposible cambiar cintas y diskettes, e incluso listados y, obviamente, cartuchos, dado que las diversas versiones de BASIC que corren en los diferentes micros son muy distintas. Basta echar un vistazo al manual de Spectrum, de Commodore o de Apple, por citar sólo algunos, para comprobar esta afirmación.

De tal modo un programa enteramente escrito en BASIC debe de ser revisado con mucha atención antes de poder adaptarlo a determinado micro. Incluso algunas marcas poseen incompatibilidades entre sus distintos modelos y en ocasiones no es posible ni siquiera aprovechar las rutinas de utilidad en lenguaje máquina, y, desde luego, ni soñar en adquirir software de un fabricante que no sea el específico.

Esta caótica situación se ha mantenido durante años debido a que ello beneficiaba a las grandes compañías ya establecidas que a través de esta avispada política comercial pretendía

perpetuar su marca en el mercado internacional. Mientras fuese la más vendida también sería más fácil encontrar software para ella y si se encontraba más software también se vendería más.

La única solución viable para los fabricantes de nuevos ordenadores hasta ahora era la emulación, la copia lisa y llana. El mundo de los ordenadores medios de gestión personal se ha visto invadido por aparatos «clónicos» (reproducciones exactas del original copiado). Así aparecieron los clónicos de Apple y posteriormente los del IBM PC el que sin embargo parece que ha logrado imponer su estándar. En el terreno de los micro ordenadores la confusión era mayor.

Es en este contexto en el que la industria japonesa de ordenadores parece lanzar un rayo de luz y lo que en un principio fue concebido como una ofensiva comercial para dominar el mercado partiendo del aserto de que la unión hace la fuerza, se ha convertido en la esperanza de un sistema estándar unificado.

Esta estandarización afecta a dos frentes:

Material, pues permite el intercambio entre aparatos de la misma norma de todo tipo de periféricos, expansiones de memoria, cartuchos, etc., sin ninguna excepción ni necesidad de adaptaciones previas.

Lógica, porque todos los programas, escritos en lenguaje máquina o BASIC-E, pueden introducirse desde cualquier periférico de almacenamiento –cartucho ROM, cassette, diskette, etc.–, a cualquier aparato de la norma sin otra limitación que la de memoria necesaria para hacerlo correr, puesto que no todos los aparatos MSX disponen de la misma memoria para el usuario. A esto hay que añadirle las amplias posibilidades de expansión de memoria, lo que favorece la configuración del sistema por parte del propio usuario.

Sin embargo, un sistema que pretende unificar el mercado necesita tener una calidad probada, que pueda mantenerse largo tiempo en marcha y gozando de buena salud. En este sentido las grandes líneas que se han seguido para el sistema MSX han sido sencillas y en ellas no se ha buscado la innovación tec-

nológica, sino su eficacia. Es decir que los fabricantes han interpretado un inteligente término medio entre un lenguaje revolucionario –el BASIC Microsoft Extended–, y un hardware clásico, que descansa sobre el microprocesador Z80 de Zilog de probada calidad y rendimiento en el mercado de los micro ordenadores.

Ahora bien, la posible creación de un estándar ha interesado tanto a usuarios como a profesionales del software. Ahora, gracias a ello, se pueden crear programas para un sistema que es apoyado por diversas marcas, lo que aumenta su posible mercado y limita los riesgos de un trabajo inútil, como en el caso de programar un ordenador fabricado por determinada firma sujeta a los avatares del mercado.

En el caso de la norma MSX se pueden producir programas para un amplio mercado con el aval de varias marcas.

Este es el motivo por el cual hay que considerar de un modo especial lo que es la filosofía MSX. En efecto, tras este sistema estándar se halla una cuidada filosofía del ordenador familiar a partir de la cual se desarrollan amplias posibilidades de interrupción, de salida para periféricos y una serie de instrucciones especializadas de su BASIC, que facilitan la integración en el sistema de todos aquellos aparatos que estos ingeniosos japoneses nos han vendido previamente –cadena HI FI, equipo doméstico, iluminación, etc.–. Bastará repasar la lista de compañías adscritas a la norma para darnos cuenta de ello. Por ejemplo JVC está trabajando en el campo del Compact Disk y el vídeo lo mismo que Sony y Pioneer, y Yamaha incorpora un interface para un teclado musical que aprovecha al máximo las posibilidades del chip de sonido incluido en las especificaciones MSX. Verdaderamente, la introducción en el mercado de las marcas japonesas marca un nuevo hito en la informática familiar, teniendo en cuenta las cotas alcanzadas en campos como el del vídeo, y el sonido, con la consecuente reducción de precios al estandarizar posiciones comerciales y ofrecer mayores prestaciones al usuario. Quizás por ello Philips, compañía europea que ofrece diversos modelos de ordenadores de distinta capacidad, ha pagado alrededor de 300.000 dólares por la licencia para fabricar un aparato MSX, aparte de los royalties por aparato vendido, y sumarse así a la arrolladora ola japonesa en el campo de la micro informática.

ASI CONCIBIO MICROSOFT SU BASIC EXTENDED

En el año 1982 Harry Fox y Alex Weiss, dos comerciantes de Nueva York dedicados a la importación de relojes desde Hong Kong, decidieron apuntarse al carro de las nuevas tecnologías de consumo y diseñaron el hardware de un ordenador que debería fabricarse en la famosa colonia británica en Asia y no superar los 50 dólares de precio de coste.

La estructura de este ordenador, que con el tiempo sería el Spectravídeo, se basaba en el microprocesador Z80 y su software de base (firmware), el producido por la firma norteamericana Microsoft.

Kaye Nishi, vicepresidente de Microsoft, fue el encargado de llevar a término el proyecto. Nishi, excelente ingeniero y avispado comerciante, había ya obtenido un rotundo éxito con el Tandy 100, aparato diseñado por él para una marca japonesa (Kyocero), y que penetró totalmente en el mercado americano, abriéndole, consecuentemente, las puertas de los EE.UU.

Pues bien, este inteligente ingeniero japonés diseñó un equipo que si bien aumentó su precio de costc a 80 dólares, disponía de expansión hasta 256 K., acceso a diversas posibilidades de display y de gráficos, un macrolenguaje musical y, lo más importante, un sistema de interrupciones fácilmente programables por el usuario que hacían posible el control de los dispositivos digitales electrodomésticos. Fue así como nació el Spectravídeo 318/328.

Nishi, que debía de sentirse muy satisfecho de su trabajo, recibió un año después –enero de 1983–, de una serie de compañías niponas el encargo de crear un sistema estándar de ordenadores personales. Obviamente el ingeniero pensó en su trabajo realizado para Spectravídeo y en marzo negoció con Weiss y Fox el permiso para fabricar el modelo. Sobre el modo en que se desarrollaron las negociaciones poco se conoce, sin embargo el 17 de junio de 1983, William Gates y Kaye Nishi, dieron a conocer en Tokio el nuevo estándar, ligeramente diferente del SV328, pero lo suficientemente similar como para que fuesen compatibles por medio de un cartucho adaptador. De todos modos, más tarde se inició la fabricación del Spectravídeo 728 totalmente compatible con la norma.

MARCA	MODELO	MEMORIA		TECLADO		CARACTERISTICAS ESPECIALES
		ROM	RAM	TIPO	N.º	
CANON	V-20	32 K	64K+16K	QWERTY	73	Interface 8 bits en paralelo p/impresoras-Buen manual de referencias BASIC
GOLDSTAR	FC-200	32K	64K	QWERTY	73	Incorpora light pen frontal
HITACHI	MB-H80	32K	64K+16K	QWERTY	73	
MITSUBISHI	ML-F80	32K	64K+16K	QWERTY	73	Incluye cinta c/curso de explicación de ordenador
PHILIPS	GV-8010	32K	48K	QWERTY	72	
PHILIPS	GV-8020	32K	80K	QWERTY	73	253 caracteres alfanuméricos accesibles por teclado normal y 5 teclados alternativos
SANYO	PHC 28P MPC-100	32K 32K	64K+16K 64K+16K	QWERTY QWERTY	72 72	Com/Bus de expansión
SONY	HIT-BIT-75 P HIT-BIT-101	32K+16K 32K+16K	64K+16K 48K	QWERTY QWERTY	74 74	Incorpora 16K d firmware Incorpora tecla de pausa y dos palancas de juegos
SPECTRAVIDEO	SVI-728	32K	64K+16K	QWERTY	73	
TOSHIBA HX-10		32K	64K	QWERTY	73	
YAMAHA CX-5M		32K	32K+16K	QWERTY	73	Incorpora el MIDI (Musical Instrument Digital Interface) y generador FM de voces
YASHICA	YC-64	32K	64K+16K	QWERTY	73	Teclado muy profesional

Los primeros micro ordenadores MSX empezaron a comercializarse el otoño del mismo año 1983 y, en poco más de un año se han vendido sólo en Japón más de medio millón de aparatos.

Ante el éxito, otras compañías japonesas, coreanas y la europea Philips se interesaron por el sistema. Al punto de que en España contamos con un importante número de fabricantes de la norma MSX que comercializan sus aparatos desde finales de 1984.

Si además de la cantidad de marcas observamos el grado de introducción en el mercado y el prestigio de muchas de ellas, podemos confiar en un futuro despejado de dudas para aquellos que adquieran un micro ordenador MSX. También es fácil suponer que el bajo precio con que se presenta el sistema MSX en el mercado, unido a la gran cantidad y variedad de software y periféricos, que puede generar debido a la compatibilidad entre marcas, revolucionará lo que se conoce como informática familiar.

No obstante el punto de mira de la competencia está puesto en el mercado norteamericano, en el que hasta el momento el estándar MSX no tiene demasiada incidencia. Es posible que los fabricantes estén esperando tener más variedad de software y periféricos acoplados al sistema para lanzarlo en EE.UU. Este mercado es difícil, aunque ya ha sido penetrado por la industria japonesa en otros campos –alta fidelidad, automotores, etc.–, y cuenta con un aliado interior, pues no debemos olvidar el Spectravídeo. Según los observadores más cualificados, el lanzamiento en Estados Unidos de la norma MSX se realizará en 1986 e irá acompañada y apoyada por una espectacular bajada de precios, que representaría sustituir la actual estructura de tarjetas LSI –circuitos integrados de gran tamaño–, por otras del tipo VLSI –circuitos de alta integración–, los cuales sustentarían toda la circuitería del sistema MSX en una sola tarjeta.

EL ORDENADOR MSX

En términos muy simples un ordenador es una máquina que obedece órdenes para efectuar cálculos y operaciones lógicas.

Para cumplir con su misión, el ordenador necesita disponer de un almacén donde guarda las instrucciones que debe ejecutar y los datos sobre los que se efectuarán los cálculos y operaciones incluidas en el programa.

Un programa es un conjunto de instrucciones con las que se alimenta un ordenador para realizar una misión determinada.

Así pues, vemos que un ordenador es una máquina abierta de ilimitadas posibilidades, que se compone de dos elementos. Una parte física, es decir la máquina en sí, a la que se denomina hardware y otra parte conceptual, representada por las instrucciones que el ordenador debe interpretar y ejecutar para realizar determinada operación. Estas instrucciones que hacen útil al ordenador se denominan software y sin él la máquina sólo es un montón de cables y circuitos lógicos sin ninguna función a realizar.

El software de un ordenador es lo que define las funciones hacia donde va dirigido. De modo que dos máquinas de idéntico hardware pueden ejecutar funciones muy diferentes, según el software con el que las alimentamos.

ESQUEMA «A»

Especificaciones Hardware MSX

- Microprocesador Zilog 80A.
- Interface programable (PPI) Intel 8255.
- Procesador de vídeo Texas Instruments 9118A ó 9928A.
- Unidad de memoria independiente para vídeo VRAM de 16 K.
- Generador programable de sonido (PSG) AY-3-8910 de General Instruments.
- Unidad MSX ROM de 32 K.
- Todos los MSX disponen de por lo menos 16 K de RAM.
- Ranura para BUS (conexión externa para accesorios).

El procesador dispone de 64 K de memoria organizada en grupos de cuatro «páginas» de 16 K. Cada una de estas páginas puede ser seleccionada de una de cuatro ranuras posibles (o es-

pacios de direccionamiento). La ranura 0 está reservada para la placa del sistema. La ranura 1 está siempre disponible como zona libre de la máquina para conectar cartuchos externos. Si se accede a las ranuras 2 y 3 conectando un módulo de expansión suministrado por el fabricante, cualquiera de las «ranuras primarias» puede ser expandida utilizando el «módulo de expansión de ranuras», dando lugar a cuatro ranuras secundarias por cada una primaria. El resultado final es un espacio de direcciones de un Megabyte en total sin incluir la Vídeo Ram (VRAM).

EL HARDWARE

La parte principal del hardware de un ordenador es la Unidad de Control.

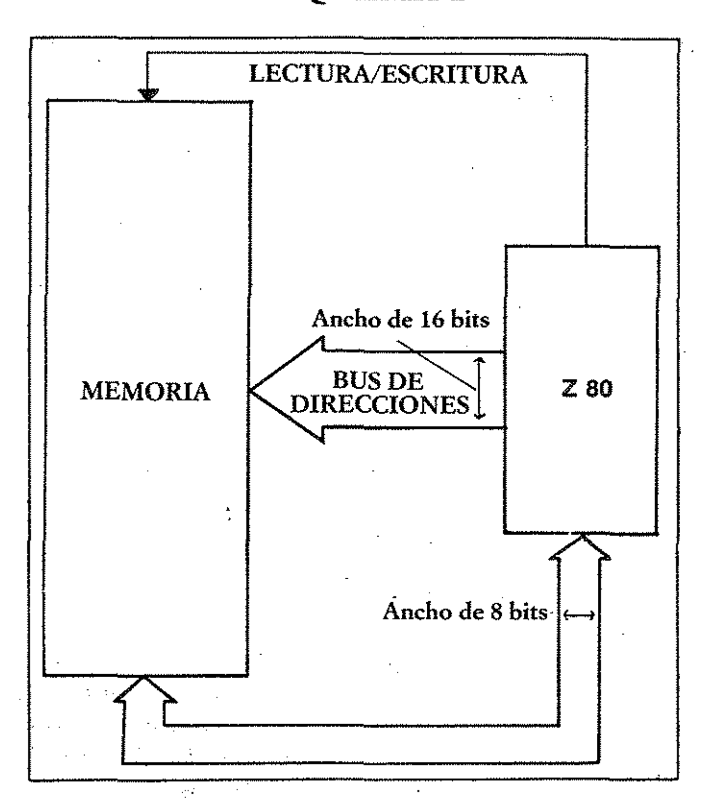
Unidad de Control. **Su misión es seleccionar las instrucciones contenidas en la memoria, decodificarlas –interpretarlas–, y hacer que se ejecuten. Si la instrucción necesita de alguna operación lógica o aritmética la Unidad de Control transfiere los datos a la Unidad Lógica y Aritmética.**

La Unidad de Control, la Unidad Lógica y Aritmética y la Memoria forman el corazón del ordenador, constituyendo el Procesador Central. Las tres partes mencionadas integran una sola placa de tecnología muy avanzada –VLSI– (Very Long Scale Integration), a la que se denomina popularmente microprocesador.

Cada modelo de microprocesador dispone de un Código Máquina peculiar que facilita el fabricante. Este Código Máquina está constituído por el conjunto de instrucciones elementales que puede entender y ejecutar el ordenador siendo en consecuencia el lenguaje del micro procesador.

Las instrucciones del Código Máquina sólo trabajan con datos binarios y mueven estos datos de la memoria a las distintas unidades del microprocesador y a los dispositivos periféricos conectados al ordenador y que comunican a éste con el exterior. Estos dispositivos realizan las funciones de recogida de datos (Input/Entrada), y la entrega de tales datos ya elaborados (Output/Salida).

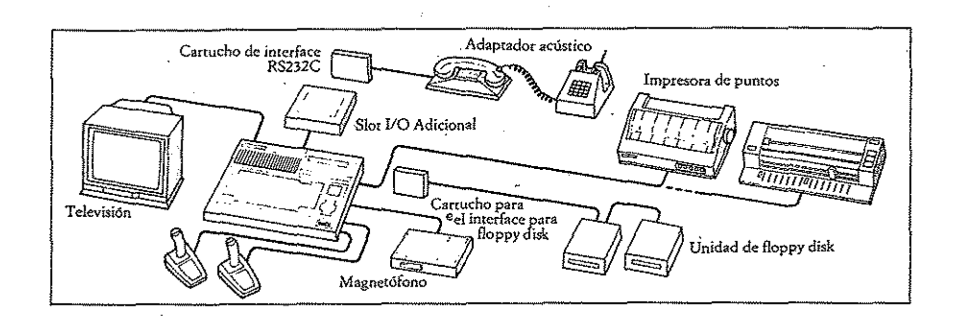
ESQUEMA 1



Los periféricos mínimos de que consta un sistema son: Teclado (entrada), pantalla de TV o monitor (salida) y cassettes para grabar o cargar programas y datos (entrada/salida). A estos suelen añadirse otros como impresora y joysticks, pizarras, cats, ligh pen y un sinfín de accesorios adaptados ya a la norma MSX.

El cassette puede ser sustituído por una unidad de diskettes que si bien cumple la misma misión es mucho más rápida para llevarla a cabo. Tanto el cassette como el diskette es una memoria de almacenamiento o memoria secundaria que conserva los datos y programas, mientras el ordenador se encarga de ejecutar otras tareas, ya que sólo puede contener un programa en su memoria central. El siguiente esquema da una idea gráfica de lo antedicho:

ESQUEMA 2



Tal como hemos señalado el microprocesador tiene su propio lenguaje y éste es lo que llamamos Código Máquina. Trabajar con él de un modo habitual es algo incómodo y engorroso, por lo que teniéndolo como base se han desarrollado otros lenguajes de más alto nivel que facilitan el trabajo del programador.

La característica más importante de estos lenguajes de alto nivel es aquella que permite que con una sola sentencia o instrucción se ejecute una gran cantidad de instrucciones en Código Máquina. Entre todos los lenguajes de alto nivel, el BASIC es quizás uno de los más desarrollados en su orientación hacia el problema. Es decir, que el programador se preocupa más de lo que quiere hacer que de cómo funciona la máquina, pues de lo contrario estaríamos ante un lenguaje orientado hacia la máquina.

En el BASIC es tal la orientación hacia la solución del problema que se suele hablar de lenguaje dialogal, es decir de un idioma que permite que el hecho de operar con el ordenador se convierte en un diálogo entre el hombre y la máquina, gracias a la variedad de instrucciones que permiten decirle al aparato que realice un mismo trabajo de formas muy diferentes. Esto es lo que hace posible que se hable de tal o cual estilo para referirse a la forma de trabajar de determinados programadores, del mismo modo que hablamos de los estilos de distintos escritores.

CAPITULO I

Construcción de programas

Llamamos programa a una sucesión lógica de pasos tendentes a realizar una misión concreta.

Estos pasos deben ser traducidos a un lenguaje inteligible por el ordenador (en este caso Basic) y ser introducidos por medio de algún periférico (teclado, lector de cassette, unidad de diskette, etc.). A partir de entonces, el ordenador lo traduce a una sucesión de números binarios con los que opera el microprocesador.

Un programa en BASIC no es sino la traducción mediante una serie de palabras clave (reservadas) de unas instrucciones concretas para trabajar con series de datos numéricos o alfanuméricos, que pueden ser fijos (constantes) o evolucionar (variables).

Más adelante veremos las variables, constantes y palabras reservadas que utiliza el sistema MSX.

Una de las grandes ventajas que nos proporciona la informática, es el hecho de permitirnos trabajar con muchos datos que varían a medida que se va ejecutando el programa. Estos datos se denominan «variables». Debemos imaginarnos estas variables como depósitos de información, ya sea numérica o alfanumérica, denominándose en este último caso cadenas.

El BASIC MSX puede ser utilizado en dos modos: directo y diferido.

En modo directo, las sentencias y comandos MSX no van precedidas por números de líneas: se ejecutan a medida que se introducen. Los resultados de las operaciones aritméticas y lógicas pueden presentarse inmediatamente en la pantalla y almacenarse para su uso posterior; pero las instrucciones propiamente dichas se borran después de la ejecución. El modo directo es útil para la corrección de programas y como calculadora en operaciones rápidas que no exijan un programa completo.

El modo diferido es lo que otros llaman modo de programa. Las líneas de programa van precedidas por sus correspondiente números, y se archivan por ese orden en la memoria. Los programas se ejecutan en respuesta al comando RUN.

Puede teclearse cualquiera línea lógica (formada por varias lineas físicas hasta un máximo de 255 carácteres) y al presionar la tecla RETURN, el ordenador examina lo tecleado y trata de realizar la tarea.

Si la línea empieza con un número, ésta se almacena en la memoria para la posterior ejecución y si ya hubiera un programa en memoria, se modifica la línea introducida sobre el programa existente. En otro caso, si no hay número de línea, se supone que se ha tecleado un comando que ha de realizarse inmediatamente. El mensaje OK, es señal de que el BASIC ha vuelto a modo directo; esto ocurre cuando se termina de ejecutar el programa, cuando se da un error en el flujo del programa, o cuando se interrumpe con las teclas de control.

Para optimizar los programas (corregir, simplificar, modificar) disponemos de un potente auxiliar: el EDITOR.

El BASIC MSX, se compone de aproximadamente 150 «expresiones» o palabras clave, las cuales se complemetan con varios subcomandos, o subórdenes que le complementan y amplían. Estas palabras clave se pueden dividir en:

COMANDOS
FUNCIONES
SENTENCIAS

Los Comandos son palabras reservadas que se utilizan para modificar algún atributo del ordenador y de su operación. Deben ser introducidos individualmente por el operador. Ejemplos de Comandos son: RUN, LIST y NEW.

Los Comandos se pueden utilizar en modo directo; las Sentencias se utilizan habitualmente en líneas de programa. No obstante, unos y otros pueden ser utilizados de forma diferente para fines específicos.

Las Sentencias son palabras reservadas que realizan la parte principal del trabajo en BASIC. Son procesadas automáticamente en orden secuencial –según número de línea– por el ordenador.

Las Funciones se usan para el tratamiento de expresiones (lógicas o aritméticas). Toman argumentos que son evaluados y utilizados luego para realizar la operación definida. El resultado de dicha operación sustituye a la función cuando es invocada en la expresión de que se trate.

En realidad las funciones enmascaran variables y constantes: por ejemplo, ABS es una función que cuando se aplica a una constante o variable la modifica dándonos su valor absoluto y, excepto raras veces, no tiene sentido si no es aplicada a un valor concreto sea constante o variable.

A continuación pasamos a detallar el conjunto de instrucciones BASIC MSX que serán tratadas en este texto. Algunas por ser típicas de cualquier BASIC, por lo tanto muy conocidas, y detalladamente explicadas en todos los manuales, no se estudiarán y nos remitimos a los programas de ejemplo para entender su funcionamiento. Otras serán estudiadas con detalle dado su peculiar interés y, por fin, las instrucciones referentes a gráficos, sonido y gestión de archivo, serán estudiadas separadamente en sendos capítulos de este libro.

ESQUEMA 3

Instrucciones MSX BASIC

ORDENES SISTEMATICAS

AUTO	– Numeración automática
CONT	– Continúa la ejecución
DELETE	– Borra líneas
LIST	– Listado de programa
LLIST	– Impresión de programa
NEW	– Supresión del programa
RENUM	– Renumeración del programa
RUN	– Ejecución del programa
TRON/TROFF	– Trazado de la ejecución

BUCLES Y RAMIFICACIONES

FOR/NEXT/STEP	– Bucle
GOSUB	– Llamada a subrutina
GOTO	– Salto a una línea del programa
IF/THEN/ELSE	– Condición y decisión
ON...GOSUB	– Bifurcación condicional
ON...GOTO	– Salto indicado
RETURN	– Fin de la subrutina

DECLARACIONES

DEFFN	– Define una función
DEFDBL	– Define las variables del tipo «doble precisión»
DEFINT	– Define las variables enteras
DEFSGN	– Define las variables del tipo «simple precisión»
DEFSTR	– Define el «tipo cadena»
DEFUSR	– Define una rutina en lenguaje máquina
DIM	– Define una tabla
ERASE	– Suprime una tabla

INTERRUPCIONES

INTERVAL ON/OFF/STOP	– Paro/funcionamiento de intervalos
ON INTERVAL GOSUB	– Interrupción tras un intervalo
ON STOP GOSUB	– Interrupción al pulsar STOP
STOP ON/OFF/STOP	– Ignora/admite la interrupción

TRATAMIENTO DE ERRORES

ON ERROR GOTO	– Salto si hay error
RESUME	– Continúa después de la corrección
ERROR	– Genera error definido por el usuario
ERL	– Número de línea del error
ERR	– Código del error

ORDENES DE USO GENERAL

CLEAR	– Borra las variables y reserva espacio para cadena y otros usos
DATA	– Datos del programa

END	– Fin del programa
INPUT	– Entrada por teclado
LET	– Asignación de variables
LINE INPUT	– Entrada de cadena
LPRINT	– Impresión
LPRINT USING	– Impresión de un formato
OUT	– Salida controlada de un bit
POKE	– Escribe sobre la memoria
PRINT	– Escritura en la pantalla
PRINT USING	– Escritura en formato
READ	– Lectura de datos
REM	– Comentarios
RESTORE	– Posiciona el puntero al inicio de los datos
STOP	– Para provisional
SWAP	– Intercambia valores entre dos variables
WAIT	– Espera llegada de un bit

FUNCIONES DE USO GENERAL

BASE	– Organiza VRAM
BIN$	– Conversión en binario
CDBL	– Conversión en doble precisión
CINT	– Conversión en enteros
CSNG	– Conversión en simple precisión
FIX	– Supresión de decimales
FN	– Función del usuario
FRE	– Memoria disponible
HEX$	– Conversión en hexadecimales
INKEY$	– Modo tablero
INP	– Entrada controlada de un bit
INPUT$	– Espera teclado
INT	– Produce un entero matemático
LPO$	– Posición del cursor impresora
OCT$	– Conversión de octales
PEEK	– Acceso de memoria (lectura)
RND	– Número aleatorio
SPC	– Imprime espacios en blanco
TAB	– Columnado del cursor
TIME	– Reloj del sistema
USR	– Conversión de una cadena en números
VARPTR	– Puntero de la posición de las variables

CALCULO

ABS	– Numeración automática
ATN	– Arco tangente
COS	– Coseno
EXP	– Exponencial
LOG	– Logaritmo neperiano
SGN	– Función signo
SING	– Seno
SQR	– Raíz cuadrada
TAN	– Tangente

TRATAMIENTO DE CADENAS

ASC	– Valor ASCII de una cadena
CHR$	– Genera un carácter ASCII a partir de número. Es complementaria a la función ASC
INSTR	– Búsqueda de subcadena
LEFT$	– Lado izquierdo de una cadena
LEN	– Largo de una cadena
MID$	– Medio de una cadena
RIGHT$	– Lado derecho de una cadena
SPACE$	– Cadena de espacios
STR$	– Conversión de un número en cadena
STRING$	– Produce una cadena

CAPITULO II

El potente editor «todo pantalla»

Cuando se introduce un programa por primera vez en la máquina, habitualmente es preciso efectuar una serie de modificaciones en su listado (intercalar alguna instrucción, modificar algún orden...). A toda la serie de recursos técnicos que nos brinda el sistema para efectuar estas correcciones lo denominamos EDITOR.

Un Editor «todo pantalla» como el que llevan incorporados los aparatos de la norma MSX es francamente útil, puesto que teniendo en pantalla el listado podemos ir a cualquier sitio que deseemos con las teclas del cursor y efectuar los arreglos pertinentes. Veamos que opciones nos proporciona este editor:

LIST: Este comando es conocido de todos los usuarios de miniordenadores; pues –como es sabido– si colocamos LIST en modo directo y pulsamos RETURN inmediatamente aparecerá en pantalla el listado del programa que esté almacenado en la memoria. El BASIC MSX enriquece este comando con una serie de variantes que lo hacen aún más útil. Si después de LIST colocamos un número de línea, tan sólo se nos presentará en pantalla la línea especificada. Si en lugar de un solo número de línea especificamos dos separados por un guión (–), serán listadas todas las líneas comprendidas entre las dos que hayamos escrito. Podemos también escribir primero el guión y después un número de línea, lo que nos proporcionará el listado del pro-

grama desde el principio hasta la línea indicada. También podemos teclear un n.° de línea y tras él el guión, lo que nos dará el listado de las líneas comprendidas entre la que hayamos escrito hasta el final del programa. En último lugar, podemos emplear un punto (.) después de LIST, con lo que obtendremos la última línea que hayamos entrado o que hayamos visualizado. El comando LIST puede también utilizarse como sentencia de programa, pero una vez ejecutada nos devolverá al modo directo. Podemos detener el listado pulsando STOP, y reemprenderlo volviéndolo a pulsar. Asimismo podemos interrumpirlo pulsando CTRL-STOP o CTRL-C. Todo lo dicho anteriormente es válido también para obtener listados por impresora, pero debemos tener en cuenta que la orden para listar por éste periférico es LLIST.

AUTO: Este comando tiene una gran utilidad. Si escribimos AUTO en modo directo y pulsamos RETURN, se inicia la autonumeración de líneas, comenzando en la n.° 10 y con incrementos de 10. También puede iniciarse la autonumeración comenzando a partir de una línea determinada y con incrementos de 10 introduciendo AUTO seguido del n.° de línea a partir del que deseemos comenzar. Podemos también modificar el incremento de líneas tecleando AUTO, seguido del incremento de líneas que deseemos; y obviamente también podemos empezar la autonumeración en una línea cualquiera y con el incremento que deseemos tecleando AUTO seguido del n.° de línea, una coma (,) y el incremento. Si en lugar de escribir un n.° de línea colocamos un punto (.), se iniciará la autonumeración a partir de la última línea escrita. Caso de que utilizando AUTO nos encontráramos con una línea que ya estuviera utilizada, seremos avisados de ello por un asterisco (*) que aparecerá al lado del n.° de línea. Si deseamos conservar la línea tal como está simplemente debemos pulsar RETURN, caso contrario borraremos el asterisco y escribiremos la nueva línea. Este comando sirve para ahorrar tiempo, puesto que nos permite ignorar los n.° de línea y para acelerar tareas repetitivas, sobre todo si combinamos su uso con el de alguna tecla de función. Para salir del modo de autonumeración debemos pulsar CTRL-STOP o CTRL-C.

RENUM: Este es un comando que nos ayuda a ordenar las líneas de un programa. Cuando hemos tecleado un programa en el que los incrementos de línea no son fijos, o cuando hemos

intercalado líneas una vez listado, etc. RENUM nos ayuda (si lo tecleamos en modo directo seguido de RETURN) renumerando todas las líneas a partir de la 10 y con incrementos de 10, actualizándose automáticamente los n.° de línea que figuren en programa en sentencias como GOTO, GOSUB, etc. Por si esto fuera poco, también se puede renumerar el programa definiendo el n.° de línea en que queremos que comience escribiendo RENUM y el n.° de línea deseado; e incluso si después de RENUM y un n.° de línea colocamos una coma (,) y un n.° de línea ya existente renumeraremos la porción del programa comprendida entre el segundo n.° de línea y el final con el n.° de línea que hayamos especificado en primer lugar después de RENUM y en incrementos de 10.

También podemos variar los incrementos, haciendo seguir a los dos valores especificados anteriormente otra coma y el nuevo valor del incremento. Si omitimos cualquiera de estos valores, debemos especificarlo mediante una coma. Unos ejemplos nos aclararán esto:

RENUM 100, 30 convertirá la actual línea 30 en línea 100, y a partir de ella todas las líneas de 10 en 10.

RENUM 100, 30, 5 hará lo mismo que el ejemplo anterior, pero los incrementos de línea irán de 5 en 5.

RENUM 7, renumerará todo el programa desde el comienzo con incrementos de línea de 7 en 7.

Si en una sentencia de programa hubiéramos utilizado un n.° de línea inexistente por error, al utilizar RENUM seríamos avisados de ello con el mensaje «Undefined line number... in...». Al margen de su gran utilidad como control del texto del programa, es muy útil también para la fusión de programas con MERGE, puesto que con RENUM podemos tener la certeza de que no se repitan números de línea. Hemos de tener en cuenta que el uso de RENUM es irreversible, por lo tanto debemos tener cuidado en señalizar convenientemente las partes del programa de interés para nosotros (subrutinas, entradas o salidas de datos...) puesto que en caso contrario podemos perder un tiempo considerable tratando de reubicarlas.

DELETE: Es sabido de todos los usuarios que para eliminar una línea de programa, basta con introducir su número y pulsar RETURN. El problema viene cuando se desea borrar un grupo de líneas, puesto que si hay muchas líneas ese procedi-

miento representa una considerable pérdida de tiempo. El Editor MSX cuenta con el comando DELETE para solucionar ese problema. Este comando puede borrar todo o parte del programa, por lo que debemos utilizarlo con especial atención. Si introducimos DELETE seguido de un n.° de línea, un guión y otro n.° de línea serán automáticamente borradas de la memoria todas las líneas comprendidas entre las especificadas, estas inclusive. También podemos omitir el segundo n.° de línea pero no el guión si deseamos borrar las líneas entre la primera y el final del programa. Asimismo, podemos colocar el guión y el 2.° n.° de línea con lo que borraremos las líneas desde el principio hasta la especificada. Si tras DELETE colocamos un punto, se borrará la última línea que hayamos visualizado. Si en el programa no existiera ninguno de los números de línea especificados en DELETE obtendríamos el mensaje «Illegal function call». De todos modos, si deseamos borrar la totalidad del programa, lo más práctico es teclear NEW con lo que además del borrado de la memoria, se reinicializan las variables, ficheros, etc.

Además de los comandos mencionados, el BASIC MSX dispone de una serie de utilísimas teclas de control y especiales que poseen la finalidad de hacer más sencillo el editaje de programas. Quedan detalladas en la figura adjunta.

Dos sentencias que van íntimamente unidas (una desactiva a la otra) también muy útiles para el editaje de programas son TRON y TROFF, que pueden ser usadas como comandos en modo directo. TRON resulta muy práctico cuando se desea saber que hace un programa durante su ejecución, es decir, cuando bifurca hacia una u otra línea o que está ejecutando en cada momento. Tras la introducción de TRON y durante la ejecución del programa se visualizarán en pantalla (siempre que estemos en un modo de texto) los números de las líneas del programa a medida que se vayan ejecutando. TROFF detiene este seguimiento.

Variantes del editor

VARIANTES DE LIST

LIST	Lista el programa completo
LIST line	Lista la línea específica
LIST ⟨line 1⟩-⟨line 2⟩	Lista el programa desde la línea 1 a la 2
LIST line-	Lista desde la línea específica hasta el final
LIST -line	Lista desde el principio hasta la línea específica

VARIANTES DE AUTO

AUTO	Numera las líneas desde el 10 en grupos de 10
AUTO line	Numera las líneas desde el número de la línea especificada en grupos de 10
AUTO line, increment	Numera las líneas desde el número de la línea especificada en grupos de incremento
AUTO, increment	Continúa desde el número de la línea que se ha dejado con el incremento especificado
AUTO line	Numera las líneas desde el número de la línea especificada con el mismo incremento que el incremento previo

VARIANTES DE RENUM

RENUM	Renumera desde el número de línea 10 en grupos de 10
RENUM ⟨line 1⟩	Renumera desde el nuevo número de la línea 1 en incrementos de 10
RENUM ⟨line 1⟩, ⟨line 1⟩, ⟨line 2⟩	Renumera desde el anterior número de la línea 2 empezando una nueva línea 1 en incrementos de 10
RENUM ⟨line 1⟩, ⟨line 2⟩, ⟨increment⟩	Renumera desde la anterior línea 2 empezando con una nueva línea 1 en el incremento especificado
RENUM, ⟨line 2⟩, ⟨increment⟩	Renumera desde la vieja línea 2, desde la nueva línea 10 con incrementos de 10
RENUM ⟨line 2⟩	Renumera desde la vieja línea 2 a partir de la nueva línea 10, con incrementos de 10
RENUM ⟨increment⟩	Renumera desde la nueva línea 10 con un incremento especificado
RENUM ⟨line 2⟩, ⟨increment⟩	Renumera empezando con la nueva línea 1 desde la vieja línea 10 con el incremento dado

VARIANTES DE DELETE

DELETE ⟨line⟩	Borra la línea especificada
DELETE-⟨line 1⟩-⟨line 2⟩	Borra desde la línea 1 a la 2 inclusive
DELETE ⟨line 2⟩	Borra desde la línea 2 inclusive

Código de teclas de control y funciones especiales

CODIGO HEX	T. DE CONTROL	T. ESPECIAL	FUNCION
02	* ⟨CTRL⟩ ⟨B⟩		Mueve el cursor hacia la izquierda una palabra.
03	* ⟨CTRL⟩ ⟨C⟩		Detiene la ejecución cuando el BASIC está esperando el INPUT. Vuelve al nivel de la orden.
05	* ⟨CTRL⟩ ⟨E⟩		Borra todo lo que está a la derecha del cursor, incluyendo el carácter que está por debajo del cursor.
06	* ⟨CTRL⟩ ⟨F⟩		Mueve el cursor hacia la derecha una palabra.
07	* ⟨CTRL⟩ ⟨G⟩		Beep.
08	⟨CTRL⟩ ⟨H⟩	BS	Retrocede y borra un carácter a la izquierda del cursor y después tira todo lo que hay a la derecha del cursor hacia la izquierda con un espacio.
09	* ⟨CTRL⟩ ⟨I⟩	TAB	Mueve hasta la próxima posición TAB. TAB está ajustada con un intervalo de 8 caracteres. Dejará blancos desde la posición de la que se ha movido.
0A	* ⟨CTRL⟩ ⟨J⟩		Alimentación de línea. Mueve físicamente el cursor hasta el principio de la próxima línea.
0B	* ⟨CTRL⟩ ⟨K⟩	HOME	Mueve el cursor hasta la parte superior izquierda de la pantalla.
0C	* ⟨CTRL⟩ ⟨L⟩	CLS	Limpia la pantalla y mueve el cursor hasta la posición de HOME (parte superior izquierda).
0D	* ⟨CTRL⟩ ⟨M⟩	RETURN	Retorno del carro. Inserta la línea en cuestión.
0E	* ⟨CTRL⟩ ⟨N⟩		Mueve el cursor hasta el final de la línea en cuestión.
12	* ⟨CTRL⟩ ⟨R⟩	INS	Se pone en modo de INSERT. El cursor se convierte en la mitad de su tamaño normal y permite insertar caracteres a la posición actual del cursor sin borrar nada.
15	* ⟨CTRL⟩ ⟨U⟩		Borra toda la línea en cuestión.
18	* ⟨CTRL⟩ ⟨X⟩	SELECT	Ignorado por la versión actual del MSX.
1B	* ⟨CTRL⟩ ⟨[⟩	ESC	Ignorado por la versión actual del MSX.
1C	* ⟨CTRL⟩ ⟨Y⟩		Mueve el cursor hacia la derecha un espacio.
1D	* ⟨CTRL⟩ ⟨]⟩		Mueve el cursor hacia la izquierda un espacio.
1E	* ⟨CTRL⟩ ⟨^⟩		Mueve el cursor hacia arriba un espacio.
1F	* ⟨CTRL⟩ ⟨_⟩		Mueve el cursor hacia abajo un espacio.
7F	⟨CTRL⟩ ⟨DEL⟩	DEL	Borra el carácter en la posición del cursor y tira la parte derecha hacia la izquierda un espacio.

La marca * indica que inhabilitará el modo INSERT si éste está puesto.

CAPITULO III

Constantes numéricas

Llamamos constantes numéricas a los números utilizados por el BASIC MSX para efectuar sus cálculos.

En el BASIC MSX, estas constantes pueden ser de seis clases:

- *Enteras:* el BASIC MSX trabaja con todos los enteros (positivos o negativos) comprendidos entre -32.768 y 32.767.
- *Decimales:* La coma decimal viene representada en MSX por un punto.
- *Notación científica:* Cualquier número demasiado grande siendo entero o demasiado pequeño siendo decimal, puede ser representado en dos partes: un número entero o decimal (mantisa) seguido de la letra E y de un entero con signo positivo o negativo (exponente). En BASIC MSX, la notación científica pérmite cálculos entre números comprendidos entre 10^{-63} y 10^{63}.

  Además de esto, los números decimales o en notación científica pueden estar expresados en simple o doble precisión.

  La precisión simple, corresponde a un almacenamiento de números hasta un máximo de 6 decimales; la doble precisión almacena hasta catorce. MSX trabaja –si no se especifica lo contrario– en doble precisión. Se puede se-

de caracteres) otra variable o constante del mismo tipo, sin sobrepasar los 255 caracteres.
Ejemplo de variable de cadena: A$ = "PEPE"

– *Variables numéricas enteras:* Su nombre debe ir seguido del signo «%» por ejemplo: A%.
– *Variables numéricas en precisión simple o doble:* Como en el caso de las constantes, su nombre debe ir seguido de ! en el caso de la precisión simple (A!) o de # en caso de doble precisión (A#); aunque por omisión, la máquina siempre trabaja en doble precisión.
De todos modos, no es preceptivo –en el caso de las variables numéricas– la colocación de signos para indicar la precisión; puesto que MSX dispone de instrucciones concretas (DEFINT, DEFSGN, DEFSTR, DEFDBL) que efectúan sistemáticamente la conversión. Los nombres de las tablas de variables siguen las mismas reglas que las variables. En MSX, 255 es el número máximo de dimensiones posibles para una tabla, y la declaración DIM sirve para fijarlas [DIMA (3, 6, 9)]. El número de elementos constitutivos de las tablas sólo viene determinado por la capacidad de memoria disponible.
Es posible convertir un número asignándolo a una variable de la precisión deseada, por ejemplo: A% = 10/3 asignará a A% el valor 3, A! = 10/3 nos dará 3.33333 y con A# (o A sola) = 10/3 obtendremos 3.3333333333333.

Las variables enteras ocupan 2 bytes de memoria, las de precisión simple ocupan 4, y las de precisión doble 8. Por lo tanto, es muy importante a la hora de ahorrar memoria o de aumentar la velocidad de ejecución de un programa, definir el tipo de variable a utilizar. Las variables de cadena, ocupan en memoria un espacio igual a la longitud de la cadena más 3 bytes.
En las tablas numéricas, cada elemento ocupa el espacio correspondiente al tipo de variable al que pertenezca la tabla, incluso en el caso de que esté vacía. Si son tablas de cadenas el espacio de memoria varía según la longitud de las cadenas que forman la tabla. Si una cadena es nula sólo ocupa 3 bytes.
Es posible, como hemos mencionado, definir una variable

leccionar uno u otro modo de la siguiente forma: si trabajamos con números decimales, para hacerlo en precisión simple añadiremos el signo ! después del número (por ejemplo 3.14!); si lo hacemos en precisión doble, añadiremos el símbolo # (por ejemplo 3.1415296#).
Si trabajamos con notación científica, E corresponde a precisión simple; D a precisión doble.

- *Hexadecimal:* En informática se usa con frecuencia el sistema de cómputo hexadecimal, dado que resulta más útil que el decimal para trabajar con bytes. En este sistema se dispone de quince signos: guarismos del 0 al 9 y letras de A a la F; siendo F = 15.
  El BASIC MSX, reconoce cualquier número como hexadecimal siempre que vaya precedido de «&H»; y automáticamente lo transforma en decimal.
- *Octal:* También es posible utilizar la base ocho para tratar los números en BASIC MSX. En este caso, disponemos de los guarismos 0 al 7 para expresar los números. Los signos «&0» sirven para identificar esta notación.
- *Binaria:* De todos es sabido que el sistema binario, es aquel en que realiza la máquina todas las operaciones. A menudo resulta más útil entrar el valor de cada bit por separado que su equivalente decimal. El prefijo «&B» sirve para que cualquier MSX identifique una sucesión de 0 y 1 como número binario.

Variables

Para definir las variables en MSX no hay limitación en cuanto al nombre asignado, pero la máquina sólo reconocerá los dos primeros caracteres. Se pueden mezclar letras y cifras, teniendo en cuenta que el primer carácter debe ser obligatoriamente una letra. Otra regla muy importante que debe observarse, es la no inclusión de palabras reservadas –en su totalidad o en parte– para nombrar las variables. En MSX existen cuatro tipos de variables:

- *Variables de cadena:* Su nombre debe finalizar obligatoriamente con el signo $ y su contenido debe ir entre comillas. Se puede asociar a una variable de cadena (o cadena

mediante la sentencia DEF seguida de INT, SNG, DBL, STR. Cuando la variable en cuestión es definida mediante una de estas sentencias no es preciso añadirle ningún sufijo para expresar su condición excepto en el caso de que se desee modificar una variable en concreto:

```
10 DEFSTR A-Z
20 C% = 3.14/ATN(1):B!=RND(1)*C%
```

En la línea 10 hemos definido todas las variables del programa como alfanuméricas, es decir, todas las letras de la «a» a la «z» son consideradas como nombres de cadenas de caracteres aunque no lleven el signo $. En la línea 20 modificamos esta definición ya que convertimos a C en variable numérica entera (%) y a B en variable numérica en precisión simple (!). Los sufijos %, !, #, $ siempre tienen prioridad sobre cualquier definición DEF.

Tratamiento de variables numéricas

El BASIC MSX nos permite utilizar un gran número de operaciones, ya sean aritméticas, Booleanas o lógicas. Pasamos a detallarlas a continuación.

OPERADORES ARITMETICOS

Los operadores aritméticos realizan operaciones matemáticas con valores cuyo orden de prioridad de ejecución es el siguiente:

^ POTENCIACION (X^Y)	eleva X a la potencia Y
– NEGACION (–X)	calcula la negación de X
* MULTIPLICACION (X*Y)	multiplica X por Y
/ DIVISION (X/Y)	divide X por Y
\ DIVISION ENTERA (X\Y)	convierte en enteros los factores de la división. Tras eso divide y el resultado también se convierte en entero.
MOD (X MOD Y)	nos da el resto de una división entera, también un entero.
+ SUMA (X + Y)	suma de X más Y
– RESTA (X – Y)	resta Y a X

OPERADORES DE RELACION

Estos operadores se utilizan para efectuar comparaciones entre valores numéricos o cadenas de caracteres. Si el resultado de la operación es verdadero se le asigna un 1; si es falso 0. Toman en cuenta –en el caso de los números– los decimales y el signo para efectuar la comparación. Si se trata de cadenas se comienza a efectuar la comparación comenzando en el primer elemento a la izquierda de la cadena y carácter por carácter. Se clasifican en el orden de los códigos ASCII, que en el caso de las letras supone que comience por las mayúsculas y siga en orden alfabético. Estos operadores son:

=	IGUALDAD	(X=Y)
< >	DESIGUALDAD	(X< >Y)
<	MENOR QUE	(X<Y)
>	MAYOR QUE	(X>Y)
<=	MENOR O IGUAL	(X<=Y)
>=	MAYOR O IGUAL	(X>=Y)

OPERADORES LOGICOS O BOOLEANOS

Estos operadores efectúan sus operaciones según las reglas del álgebra de BOOLE. Pueden utilizarse para comparar valores numéricos o junto con operadores de relación. Trabajan de la siguiente manera:

NOT		EQV			AND		
X	NOT X	X	Y	X EQV Y	X	Y	X AND Y
1	0	1	1	1	1	1	1
0	1	1	0	0	1	0	0
		0	1	0	0	1	0
		0	0	1	0	0	0

IMP			OR			XOR		
X	Y	X IMP Y	X	Y	X OR Y	X	Y	X XOR Y
1	1	1	1	1	1	1	1	0
1	0	0	1	0	1	1	0	1
0	1	1	0	1	1	0	1	1
0	0	1	0	0	0	0	0	0

Código de caracteres ASCII

El MSX tiene guardados en ROM una serie de caracteres que utiliza para comunicarse con el usuario por medio del teclado. Estos caracteres comprenden las letras, los números, los caracteres especiales como asteriscos, paréntesis, etc. etc.

A cada uno de estos caracteres le corresponde un número de código. Este número de código es estandard respecto a los caracteres alfanuméricos y especiales y fue definido por una serie de fabricantes americanos, que dejaron un gran número de códigos libres, para que cada fabricante los utilizara a su estilo. Es por eso por lo que el MSX, puede disponer de una gran serie de caracteres gráficos, que pueden obtenerse directamente pues están definidos también en ROM y su forma de aparecer en la pantalla responde a las llamadas de teclado, como cualquier letra o símbolo común. Estos caracteres aparecen mediante la combinación de las letras ordinarias del teclado más la tecla GRAPH, o con MAYUSCULAS (SHIFT) más GRAPH más letra ordinaria, etc. etc.

Los números de código, pueden ser utilizados directamente por el programador por medio de las funciones ASC y CHR$.

A = ASC (C$)

Esta función almacena en A el código ASCII del primer miembro de la cadena C$. Asi `10 PRINTASC("Pepe")`

A$ = CHR$(A)

De modo que crea una cadena A$ de un carácter, correspondiente al código ASCII incluido en el paréntesis.

```
10 PRINTCHR$(80)
10 PRINTCHR$(1);CHR$(65)
```

Los números de código del ASCII se reparten de la siguiente manera: los primeros 31 códigos (del &H00 al &H1F) están reservados para el sistema, que los utiliza para gestionar la pantalla, el editor de toda-pantalla, etc. etc. Estos 31 códigos están detallados en la tabla n.° 3 junto con sus equivalencias en decimal y hexadecimal, y corresponden en la **tabla 1** a los espacios en blanco que se encuentran entre H00 y H1F. La forma en que estos caracteres se pueden ejecutar, es mediante una instrucción PRINT, es decir, si le pedimos al ordenador que ejecute tanto en modo directo como diferido el mandato PRINT CHR$ (12), el efecto que observaremos será la ejecución de CLS (borrado de pantalla); de la misma forma, si ejecutamos un PRINT CHR$ (7) oiremos un BEEP (ver capítulos dedicado al sonido).

A continuación de estos 31 códigos nos encontramos con la serie que va desde el 32 al 255, que incluyen una serie de caracteres (mayúsculas, minúsculas, teclas como ¿ ? & (/ etc. etc. y hasta un alfabeto griego).

Sin embargo lo más interesante para el programador que desea realizar juegos los códigos que más le interesan son los códigos gráficos que se encuentran ubicados en dos bloques. El primero que consta de 32 caracteres va desde el código 192 al 223 (HC0 y HDF) lo que puede comprobarse en la **tabla 1**, la que nos da los códigos ASCII en Hexadecimal. De esta manera ahorramos espacio y el usuario puede fácilmente, con ayuda de su ordenador transformarlos en decimal. Por otra parte el sistema no tiene ningún inconveniente en aceptar un PRINT CHR$(&HC0).

El esquema 2 nos muestra la segunda serie de 32 caracteres gráficos, cuyos códigos están por encima del 255 que es la máxima expresión que podemos darle a la función CHR$, por lo tanto tendremos que utilizar un truco, que consiste en realizar un PRINT CHR$(1), que equivale a oprimir la tecla GRAPH, o sea que si acto seguido mediante un punto y coma (;) hacemos

un PRINT CHR$(65), nos da un sol sonriente de color blanco, o sea PRINT CHR$(1);CHR$(65), pero hay que tener cuidado, porque este truco sólo funciona con punto y coma. Una coma sola, o dos PRINT consecutivos no producirán el efecto deseado. En la tabla 2 encontraremos los códigos de esta segunda serie de caracteres en hexadecimales que desde el &H40 hasta el &H5F y pueden utilizarse directamente de esta forma: PRINT CHR$(1); PRINT CHR$(&H41), lo que produce el mismo efecto que en el ejemplo anterior.

La utilidad de poder controlar estos códigos ASCII en nuestros programas es grande, pues de esta forma podemos comparar cadenas de caracteres, y ordenarlos por su código que coincide con el orden alfabético. Aparte, nos permite todo tipo de cálculos aritméticos con el valor de los códigos, pero esto, es mejor ir aprendiéndolo con la práctica, a base de copiar y analizar programas y, lo más importante, practicando con tu ordenador.

Para obtener el código hexadecimal de un carácter determinado, en primer lugar debemos localizar el carácter, y la numeración que encontraremos en la parte superior, en sentido horizontal, nos dan el primer dígito de dicho carácter. El segundo lo obtenemos a la derecha de la **tabla 1** (a la izquierda en la tabla 2) de esta manera la A mayúscula nos da el código &H41. La **tabla 2** funciona de la misma forma con la peculiaridad de que en la fila superior, sólo tenemos dos números el 4 y el 5, lo que es lógico, pues sus valores van del &H40 al &H5F. Por ejemplo el corazón se obtiene en pantalla mediante PRINT CHR$(1); CHR$(&H43).

TABLA 1

1.° CARACTER / 2.° CARACTER

2.° \ 1.°	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	DESDE 00 A 1F			0	@	P	`	p	Ç	É	á	Ã	▁	◀	α	≡
1			!	1	A	Q	a	q	ü	æ	í	ã	▚	⧗	β	±
2			"	2	B	R	b	r	é	Æ	ó	Ĩ	█	⧓	Γ	≧
3			#	3	C	S	c	s	â	ô	ú	ĩ	█	▪	Π	≦
4			$	4	D	T	d	t	ä	ö	ñ	Õ	▬	▐	Σ	⌠
5			%	5	E	U	e	u	à	ò	Ñ	õ	█	▐	σ	⌡
6			&	6	F	V	f	v	å	û	ª	Ũ	▌	▄	μ	÷
7			'	7	G	W	g	w	ç	ù	º	ũ	▞	▩	γ	≈
8			(	8	H	X	h	x	ê	ÿ	¿	Ĳ	█	∆	Φ	○
9			)	9	I	Y	i	y	ë	Ö	⌐	ĳ	▐	‡	θ	●
A			*	:	J	Z	j	z	è	Ü	¬	¾	█	ω	Ω	-
B			+	;	K	[	k	{	ï	¢	½	~	╱╱	█	δ	√
C			,	<	L	\	l	¦	î	£	¼	◇	╲╲	▄	∞	ⁿ
D			−	=	M	]	m	}	ì	¥	¡	‰	▼	▌	φ	²
E			.	>	N	^	n	~	Ä	Pt	«	¶	▲	▐	ϵ	▌
F			/	?	O	_	o	△	Å	ƒ	»	§	▶	▀	η	

TABLA 2

2.° \ 1.°	4	5
0		┼
1	☺	┴
2	☻	┬
3	♥	┤
4	♦	├
5	♣	┼
6	♠	│
7	•	─
8	◘	┌
9	○	┐
A	◙	└
B	♂	┘
C	♀	╳
D	♪	╱
E	♫	╲
F	☼	┼

TABLA 3

Código decimal	Código hexa.	Concepto
1	1	Vacío (interpreta el siguiente como gráfico)
2	2	Pone el cursor al inicio de la siguiente palabra
3	3	Interrumpe el programa
4	4	Vacío
5	5	Borra la línea hacia la derecha
6	6	Como el nº 2
7	7	Equivale a BEEP
8	8	Equivale a BS
9	9	Equivale a TAB
10	A	Baja el cursor una línea
11	B	Equivale a HOME
12	C	Equivale a CLS
13	D	Equivale a RETURN
14	E	Mueve el cursor al final de la línea
15	F	Vacío
16	10	Vacío
17	11	Vacío
18	12	Equivale a INS
19	13	Vacío
20	14	Vacío
21	15	Borra la línea donde está el cursor
22	16	Vacío
23	17	Vacío
24	18	Vacío
25	19	Vacío
26	1A	Vacío
27	1B	Vacío
28	1C	Equivale a cursor derecho
29	1D	Equivale a cursor izquierda
30	1E	Equivale a cursor arriba
31	1F	Equivale a cursor abajo

CAPITULO IV

Series, tablas y cadenas

Muchos problemas de programación hacen necesario el procesamiento de diversas variables en forma idéntica: en estos casos es interesante crear una serie, es decir, una determinada cantidad de variables que tienen un mismo nombre y se distinguen unas de otras por su subíndice (un número asociado al nombre de la variable). Este subíndice se escribe entre paréntesis y a continuación del nombre de la variable; de esta manera, A(0), A(1), A(2), A(3), etc. etc.

Los subíndices de las series van desde 0 hasta 255 y pueden ser perfectamente variables; A(Z); BT(J); A$(P), etc.

En un mismo programa se pueden emplear nombres de matrices idénticos a nombres de variables simples, es decir, el sistema distinguirá entre A, A$, A(n) y A$(n). El sistema asigna automáticamente 11 espacios en su memoria (no confundir con áreas de memoria) para los subíndices de 0 a 10, lo cual quiere decir, que no necesitamos definir de ninguna manera especial las tablas de menos de 11 elementos como veremos en este ejemplo.

```
10 FOR I=0 TO 10
20 READ A$(I)
30 NEXT
40 PRINT A$(4);A$(1);A$(0)
50 DATA na,cio,30,40,fun,60,70,90,100,11
0,120

        funciona
```

Sin embargo, para series mayores de 11 elementos, es necesario utilizar previamente una instrucción DIM cuya sintaxis es DIM A(n), o DIM A$(n) donde A puede ser una variable numérica de precisión sencilla, doble o entera y n es un subíndice de 1 a 255 (un subíndice 0 nos daría una variable simple) a continuación podemos, tras una coma, dimensionar otras series.

DIM A(n), B$(n), Z(n)

La utilización de series combinadas con bucles, ahorra mucho trabajo de escritura y posibilita gran número de algoritmos útiles en programación.

En ocasiones será útil crear una tabla que no es sino una serie con dos o más subíndices, como A(n,m) lo cual nos dará una serie de elementos que localizaremos por medio de dos números, el (1,5) el (2,6), etc. etc. Estos dos números nos apuntan hacia un solo contenido, dentro de la tabla de variables que responde a un nombre concreto.

Con el esquema representa una tabla de 3 por 3 elementos, o sea, una tabla del tipo

DIM A(2,2)

A(0,0) A(1,0) A(2,0)
A(0,1) A(1,1) A(2,1)
A(0,2) A(1,2) A(2,2)

De la misma forma que en las series, las tablas con subíndices menores de 10 no necesitan ser dimensionadas.

Sin embargo, sería muy útil si deseamos ahorrar memoria dimensionar las tablas y series aun cuando sus subíndices sean menores de 10, pues de esta forma el sistema reserva, justo la memoria necesaria. Es decir, si creamos una tabla del tipo DIM (2,3) el sistema sólo reserva el espacio necesario y ninguno más, no como en el caso de omitir la sentencia DIM, pues entonces por omisión reservaría una serie de espacios (121) equivalente a un DIM (10,10).

Es posible crear series y tablas de cualquier tipo mientras que este tipo sea declarado. Por ejemplo: DIM S$(100) dará una serie de 101 elementos de cadena.

Cuando se inicializa una orden, todos los valores son asumidos como cero para series y tablas numéricas y cadenas nulas para series de cadenas.

```
10 SCREEN 0:CLS:KEYOFF
20 LOCATE 7,10
30 COLOR 1,12
40 PRINT"ESTE PROGRAMA DEMUES-"
50 PRINT"TRA QUE NO ES NECESARIO DIMEN-"

60 PRINT"SIONAR UNA TABLA DE HASTA 121"
70 PRINT"ELEMENTOS."
80 FOR A=1 TO 3000:NEXT
90 SCREEN 1
100 FOR P=1 TO 6
110 FOR I=1 TO 2
120 READ P(P,I)
130 NEXT I,P
140 PRINT:PRINT:PRINT:PRINT"LOS DATOS Y
SUS CUADRADOS     ESTAN ORDENADOS POR PAR
ES"
150 LOCATE 0,10:FOR P=1 TO 6:PRINTTAB(3)
,P(P,1);TAB(15)P(P,2)
160 NEXT
170 DATA 5,25,3,9,4,16,6,36,7,49,9,81
```

Requerimientos de memoria para series y tablas

Las variables que utilizamos en nuestros programas BASIC son almacenadas en una zona de memoria llamada DIM AREA, VARIABLE AREA o STRING AREA según sean tablas variables numéricas o cadenas, ya vimos que según el tipo de variable que utilizáramos (numérica entera, de precisión doble, o cadena) necesitaban más o menos bytes para ser almacenadas. En el caso de las tablas, si son de números enteros es necesario reservar los bytes por elemento,

las de precisión simple 4 bytes por elemento

las de doble precisión 8 bytes por elemento

y las cadenas 3 más la longitud del contenido por cada elemento.

Mediante la función VARPTR puedes encontrar la situación de cada una de las variables utilizadas, ya sean cadenas, series tablas, variables numéricas. Ejemplo

```
10 A% = 100
20 PRINT VARPTR (A%)
```

Todo el contenido de las variables de cadena está almacenado en la sección STRING AREA O AREA DE CADENA de la memoria del ordenador (ver mapa de memoria). El tamaño de este String área es de 200 bytes, por defecto, es por ello que la máxima cantidad de elementos de una cadena es de 200 caracteres, si no incrementamos este área de cadenas mediante la instrucción CLEAR lo que será necesario, casi siempre que utilicemos una tabla de cadenas o una serie de cadenas de una longitud.

Utilización de la sentencia CLEAR

Con CLEAR limpiamos el espacio de memoria para variables y anulamos todas las variables de cadena y numéricas que hayamos creado y almacenado, previas a la ejecución de esta sentencia. Consta de dos parámetros, con el primero controlamos el espacio disponible para cadenas (200 Bytes por omisión) y lo podemos incrementar. Por ejemplo CLEAR 255 aumentará el área de cadenas a 255 bytes.

Es necesario tener en cuenta que tras una orden CLEAR, también se cierran todos los ficheros o dispositivos de archivo que pudieran estar abiertos.

El segundo parámetro, tras una coma (,) reserva espacio para programas en código máquina, por lo que no será tratado aún.

En ocasiones puede ser muy útil conocer el espacio que nos queda libre en memoria para introducir más líneas de programas, y de forma particular, conocer el espacio libre de que disponemos en el área de cadenas (string area). Todo esto podemos averiguarlo mediante la función FRE que posee dos sintaxis distintas. Con FRE (0) nos da el número de bytes libres que quedan en memoria.

PRINT FRE (0) recibirá una respuesta automática del sistema, ya sea en modo directo o diferido. Otra aplicación seria

```
25 IF FRE(0) < X THEN 50
```

con lo que en caso de que dispongamos de menos de X bytes de memoria libre nuestro programa transfiere el control a la línea 25.

Con la siguiente sintaxis FRE (" ") el sistema nos entrega sólo el espacio libre para cadenas, lo que es vital, pues en el sistema MSX el espacio para cadenas es fijo a lo largo de la ejecución de una tarea. Por lo tanto sólo podemos dimensionar al

principio de programa mediante CLEAR, pues cualquier sentencia CLEAR a mitad de programa nos destruirá las variables acumuladas, por ello es importante mediante FRE (" ") controlar el espacio libre para introducir nuevos datos, y si no es posible seguir introduciendo, enviarlo al fin de ejecución.

IF FRE (" ") < 20 THEN END,

Introduciendo en el lugar conveniente esta línea en nuestro programa para tratamiento de cadenas, cuando no tengamos más de 20 bytes libres en el STRING AREA (área libre para cadenas), el programa terminará.

Tratamiento de cadenas

Ya hemos visto que denominamos cadenas literales a las variables, que pueden contener caracteres alfanuméricos, tanto números como letras. En el caso de que estas cadenas contengan números, estos no tendrán valor aritmético, sino simplemente gramatical, es decir que si definimos una variable como cadena mediante una asignación A$ = "xxxxx" esta puede contener letras, números y caracteres especiales (gráficos o de control) introducidos por medio de sus códigos.

Aunque el contenido de una cadena sea únicamente numérico, la única operación que podemos ejecutar con ella es la concatenación con otra cadena; según podemos apreciar en el siguiente ejemplo:

```
10 A$= "123"
20 B$= "456"
30 LPRINT A$+B$
```

Después de listar este corto programa no da como resultado 123456.

Recordemos al ver estos ejemplos, que una cadena de caracteres se define siempre con un símbolo $ detrás de su nombre o con una declaración previa mediante la instrucción STR. Si cambiamos en la línea 10 y 20 del anterior programa los valores de A$ y B$, y los sustituimos por los valores que deseemos (cualquier carácter hasta un límite de 255) podemos ver cómo

funciona esta operación de concatenación. Por ejemplo si A$ ="ABC" y B$ ="DEF" el resultado será TOTAL="ABCDEF".

Si a continuación creamos una cadena nula, o sea, una cadena cuyo contenido sean uno o varios espacios, podemos concatenarla con otra u otras, de manera que deje espacios en blanco entre los contenidos de otra cadena. Por ejemplo:

```
10 A$=" "
20 B$="a"
30 PRINT A$+B$+A$+B$
```

Esta posibilidad es muy socorrida a la hora de manipular archivos en cassette.

A continuación veremos aquellas instrucciones que se refieren al tratamiento de cadenas:

LEFT$

SINTAXIS: A$=LEFT$(C$,N)

La instrucción LEFT$ crea una subcadena constituida por los N primeros caracteres extraídos por la izquierda de una cadena C$. Si N=0, el resultado de la operación es una cadena nula " ", si pór el contrario N es superior al número de caracteres contenidos en C$, A$ contendrá la totalidad de C$

Ejemplo:

```
10 PRINT LEFT$ ("Javier",3)
Jav
```

RIGHT$

SINTAXIS: A$=RIGHT$(C$,N)

Esta función realiza la misma operación que LEFT$, pero empieza a extraer caracteres por la derecha (evidentemente RIGHT es derecha en inglés y LEFT izquierda).

Ejemplo:

```
10 PRINT RIGHT$("Javier",2)
er
```

MID$

SINTAXIS: A$=MID$(C$;N,M)

Extrae para crear la nueva subcadena, N caracteres a partir del número N inclusive, empezando por la izquierda. Si se omite N, o si su valor es superior al total de caracteres desde el número N hasta el fin, la función envía todos estos, es decir, que por omisión de N, la nueva cadena contiene los caracteres desde N hasta el fin.

Naturalmente, si N es superior a la longitud de A$ el resultado será cadena nula.

Ejemplo:

```
10 PRINT MID$ ("Javier",2,1)
a
10 PRINT MID$ ("Javier",2,2)
av
10 PRINT MID$ ("Javier",3)
 vier
```

LEN$

SINTAXIS: A$=LEN$(C$)

Nos dará en A el número de caracteres (incluidos los no imprimibles y de control, de que consta la cadena C$.

Si C$ es una cadena nula A será 0.

Ejemplo:

```
10 PRINT LEN ("JAVIER")
 6
```

STRING$

SINTAXIS: A$=STRING$(N,X)

Con esta función creamos una nueva cadena A$ conteniendo N veces el valor del primer carácter de una cadena C$, o el carácter correspondiente al código ASCII del número X.

```
10 PRINT STRING$(5,"PEPE")
PPPPP

10 PRINT STRING$ (5,&HA5)
ÑÑÑÑÑ
```

SPACE$

SINTAXIS: A$=SPACE$(N)

Crea una cadena con N espacios (caracteres en blanco).

```
10 FOR N=1 TO 3
20 PRINT SPACE$ (N);" TEXTO"
30 NEXT

                  TEXTO
                   TEXTO
                    TEXTO
10 PRINT SPACE$ (15);"TEXTO"
                    TEXTO
```

INSTR

SINTAXIS: A$=INSTR(N,A$,C$)

Esta función busca en una cadena A$, la cadena o subcadena C$, a partir del carácter número N, o del primer carácter si se ha omitido N. El resultado que se acumula en A, es un valor numérico, indicando el número de carácter a partir del que encontremos la subcadena C$, comenzando por el primer carácter de la izquierda, tanto si lleva el parámetro N como si no, pues la introducción de este parámetro es a efectos de acelerar el proceso si se conoce la posible ubicación de la subcadena, a buscar dentro de la cadena principal.

Si la subcadena C$ no se encuentra, el valor obtenido es 0.

Ejemplo demostrativo

```
10 PRINTINSTR ("ABCDEFGHI","FG")
 6

10 LPRINTINSTR (7,"ABCDEFGHI","FG")


   0

10 PRINTINSTR (1,"ABCDEFGHI","AB")
 1
```

STR$

SINTAXIS: A$=STR$(N)

Sabemos que cuando dos números son almacenados en variables del tipo A,B etc o A%,B% o bien A!,B! o por último A#B#estamos hablando de valores aritméticos y matemáticos, pero si esos dos números se almacenan en una variable de cadena A$;B$ sólo contará para nosotros su valor gráfico o gramatical, y además de tener que tratarlos entre comillas "200", "300" no podremos, con ellos ejecutar más que la operación de concatenación. Esta función transforma una variable numérica en una de cadena, por ejemplo:

```
10 A=300 :B=200
20 A$=STR$(A)
30 B$=STR$(B)
40 PRINT A+B
50 PRINT A$+B$
 500
 300 200
```

El resultado será 500 en el caso de varibles numéricas y 200, 300 en el de cadenas. Obsérvese el espacio que precede a cada valor alfanumérico, que corresponde al signo positivo, si el número lo es. Si es un número negativo el primer espacio antes de los valores numéricos no se quedará en blanco, sino que será rellenado por el carácter alfanumérico.

Ejemplo

```
10 A=-300 :B=-200
20 A$=STR$(A)
30 B$=STR$(B)
40 LPRINT A+B
50 LPRINT A$+B$
-500
-300-200
```

VAL

SINTAXIS: A$=VAL(AS$)

Esta función es la inversa de la anterior y nos da el equiva-

lente numérico de una cadena de caracteres. Sin embargo esta conversión es más compleja.

Si el contenido de A$ es un número, el transporte a la variable A es sencillo, se archiva el número, con el espacio reservado para el signo, pero si A$ contiene números y caracteres, la función sólo archivará en A el número que se encuentre en primera posición de la cadena. Si el contenido de A$ no puede ser evaluado por empezar la cadena con un carácter alfabético, el valor archivado será 0. Para finalizar diremos que para la función VAL los espacios no tienen sentido, no los aprecia. Los prefijos de conexión entre bases &B, &O, &H son perfectamente interpretados por VAL.

Naturalmente, vistas de esta forma las instrucciones se presentan claras, pero, poco útiles. Nada más lejos de la realidad. Bastará entremezclarlas y operar con ellas dentro de series de operaciones complejas que impliquen varias de estas operaciones unidas, para darnos cuenta de la dificultad que representa trabajar con este tipo de operaciones: por ejemplo analicemos la siguiente expresión:

```
10 A$="300"
20 A=VAL(A$)
40 PRINT A/2
RUN
 150
10 A$="3 CORDEROS "
20 A=VAL(A$)
40 PRINT A
RUN
 3
```

Sobre su utilidad hay que señalar, especialmente si usted está interesado en programar para la gestión, que analice los programas del libro en que aparecen operaciones con variables de cadena, y se convencerá no tan sólo de su utilidad sino también de su necesidad.

CAPITULO V

Grabación de programas

Lo primero que hay que hacer para efectuar una grabación es verificar que las conexiones estén realizadas correctamente. Con cada MSX se facilita un cable especial para conectar al magnetófono y este cable tiene en uno de sus extremos una clavija DIN, que debe insertarse en la entrada TAPE de la parte trasera del ordenador. En el otro extremo del cable hay tres clavijas; una blanca, que debe insertarse en la toma AUDIO (EAR) del cassette, una roja, que debe conectarse en MIC, y una más pequeña negra que debe conectarse en REMOTE. En el caso de que el magnetófono no disponga de REMOTE, esta clavija cuya función es poner en marcha o detener el magnetófono manipulándolo desde el ordenador, puede dejarse sin conectar.

Una vez realizados estos pasos previos, el cassette y el ordenador ya están en situación de **dialogar.** De todos modos, es preciso tener en cuenta otra serie de detalles que pasamos a mencionar.

- Utilice siempre un cassete monoaural. Con un estéreo sólo grabará parte de la información.
- El volumen del cassette debe estar alto.
- El tono debe estar situado en agudo.
- El cabezal del cassette debe estar limpio y en perfectas condiciones.

– Azimut correcto. El azimut o grado de inclinación del cabezal de lectura, debe ser siempre paralelo a la cinta del cassette. La calidad de reproducción depende en gran medida de la correcta posición del azimut. Caso de no estar bien regulado podemos encontrarnos con que no lea exactamente la cara correspondiente de la cinta sino que lea parte de la otra cara. También ocasiona problemas de carga debido a que lee en sentido inverso parte de lo grabado en la cara opuesta. Para graduar el azimut de un cassette basta con regular con un destornillador el pequeño tornillo que hay detrás del cabezal de lectura. De todos modos, si no te atreves a hurgar en tu cassette, otra solución más cara pero igualmente válida es grabar las cintas por una sola cara.

Las grabaciones efectuadas en un magnetófono y reproducidas en otro también pueden dar problemas. Entre uno y otro aparato suele haber pequeñas diferencias de velocidad que afectan considerablemente la grabación y reproducción de programas. Lo ideal es utilizar siempre el mismo cassette.

Para la transmisión de datos del ordenador al cassette disponemos de varias alternativas: CSAVE, SAVE y BSAVE las que veremos más detalladamente en el capítulo de gestión de archivos. Bástenos saber de momento CSAVE es la orden más usual para grabar programas en cassette. Se utiliza de la siguiente manera:

CSAVE «nomprog»

El nombre del programa no debe sobrepasar los seis caracteres y el primero de ellos debe ser preceptivamente una letra. Una vez puesto el magnetófono en grabación se pulsa RETURN y se espera a que el ordenador nos muestre OK en la pantalla, cosa que significará que ha efectuado la grabación.

A pesar de que nos haya aparecido OK en pantalla, es conveniente efectuar una comprobación de la grabación. Para ello –si tiene el REMOTE conectado– teclee MOTOR ON o tan sólo MOTOR y rebobine la cinta hasta la posición donde ha iniciado la grabación.

Si conectamos la clavija REMOTE al magnetófono y ponemos éste en marcha, con la orden MOTOR ON comenzará a funcionar el cassette y con MOTOR OFF se parará. Si tan sólo

tecleamos MOTOR y el cassette está funcionando se parará y si está parado comenzará a funcionar.

Otra posible utilidad de la orden MOTOR es –una vez desconectada la clavija de audio(EAR)– conectar la puesta en marcha y la parada de una grabación AUDIO mientras se va oyendo. Esta posibilidad, tal vez un poco peregrina, la comentamos por su posible utilidad para mecanografiado de textos, o también para su uso en programas educativos que utilicen voz mezclada con programa.

Tras poner el cassette en PLAY escriba CLOAD?, pulse RETURN y el ordenador verificará que la grabación se haya efectuado correctamente mostrando un OK en la pantalla.

Cargar un programa en el ordenador también es muy sencillo: Una vez colocada la cinta deseada en el cassette se introduce CLOAD «nomprograma» y el ordenador va leyendo la cinta hasta llegar al programa deseado. Caso de no especificar el nombre del programa, el ordenador cargará por defecto el primer programa que encuentre en la cinta. Si tenemos una duda con respecto a los programas que hay grabados en una cinta, podemos verificarla totalmente con CLOAD si ponemos un nombre de programa ficticio, ya que CLOAD nos irá informando de los nombres de los programas que va encontrando en su camino... Si se produjera alguna anomalía durante las operaciones de grabación o carga del programa, la máquina nos informaría de ello con el mensaje «Device I/0 error».

CAPITULO VI

Gestión de archivo y grabación de datos

El sistema MSX para su gestión intensa considera la existencia de cuatro dispositivos de archivo periféricos con los cuales intercambia información.

1 PANTALLA EN MODO DE TEXTO (sólo de salida)
2 PANTALLA EN MODO GRAFICO (sólo de salida)
3 MAGNETOFONO A CASSETTE (entrada/salida)
4 IMPRESORA (sólo salida)

Los nombres que reciben estos dispositivos con respectivamente: CRT, GRP, CAS, LPT. Todas las salidas de información se realizan a través de estos dispositivos.

Dispositivo cassette

Recordemos que para almacenar un programa en cassette ejecutábamos en modo directo, CSAVE «PROGRA» (Donde PROGRA es el nombre del archivo que contiene el programa). Esta instrucción se complementaba con CLOAD (para carga de programas, recuerde que el cassette es de entrada/salida). Pues bien, estas instrucciones pueden ser sustituidas por SAVE «CAS:PROGRA» y LOAD«CAS:PROGRA», que sería la forma general para tratar la entrada o salida por los dispositivos archivadores.

Sin embargo, el formato en que graba los datos la orden

CSAVE es distinto del que utiliza mediante SAVE"CAS: (nombre de archivo)." En el primer caso se almacena la información en un sistema intermedio, sólo útil para leer o cargar programas. En el segundo se graba en ASCII puro, lo que resulta muy útil para trabajar con textos y datos.

Una de las peculiaridades de CSAVE y CLOAD es que no necesitan parámetros, como SAVE o LOAD, y por tanto son más sencillos de utilizar, pero el inconveniente es que no permiten efectuar MERGE de programas, pues para utilizar esta sentencia es necesario que los archivos de carga se hallen grabados en ASCII.

Con la instrucción MERGE "CAS": (nombre programa)" unimos dos archivos, el llamado desde la instrucción que se carga de cassette y el residente en memoria viva (RAM), es decir, que sólo se podrán fusionar programas grabados mediante SAVE.

El programa residente seguirá tal cual y el miembro de carga se fusionará con él, sin embargo si se repiten números de líneas éstos serán modificados en el ordenador, prevaleciendo las líneas del programa que se ha leído mediante MERGE. Si se omite el nombre de archivo el ordenador intentará fusionar el primer archivo que encuentre.

Otras instrucciones especiales que pueden ser utilizadas con el dispositivo cassette (CAS) son:

BSAVE, que graba datos en formato binario.

BLOAD, que los carga en binario (B de binario delante de SAVE o LOAD).

Estas dos órdenes están diseñadas de cara a la grabación de programas en lenguaje máquina, por lo que disponen de algún parámetro del que no disponían CSAVE y CLOAD. La instrucción completa queda como sigue:

BLOAD "nombre de dispositivo: (nombre de archivo)", R, (dirección de memoria)

Si deseamos omitir (incluso los dos puntos), nos cargará lo primero que encuentre. Los siguientes dos parámetros también son optativos. Con la R a continuación, el programa se ejecuta automáticamente tras la carga, lo cual es muy útil, pues, nos evita trabajar con las áreas de memoria —condición necesaria para ejecutar un programa en lenguaje máquina mediante las órdenes USR y DEFUSR.

Por fin el último parámetro también opcional, nos permite introducir un valor de partida que es añadido a las direcciones inicial, final y de ejecución, que se indicaron en la orden CSAVE, para posicionar en otro sitio la rutina que se carga. Por ejemplo: BSAVE "CAS:(nombre de archivo)", (dirección inicial), (dirección final), (dirección de ejecución).

En este sistema de almacenamiento, después de introducir el nombre del archivo, se deberán introducir las direcciones de memoria, inicial y final (las áreas de memoria) entre las que se encuentra la rutina en código máquina, para que sólo guarde la información contenida en estas áreas. El último, parámetro optativo nos indica la sección del programa en máquina, en que empezará la ejecución si se carga con R en la sentencia BLOAD. Si se omite este parámetro, empezará la ejecución en la dirección inicial, lo cual resulta ser lo más común.

Igualmente se puede omitir el nombre del archivo. Si así se hiciera la instrucción grabaría una cadena nula.

LOAD sólo puede asignarse a cassette pero SAVE puede utilizar los cuatro dispositivos.

La sentencia SAVE aplicada a un dispositivo CRT produce el efecto de LIST. Si se ejecuta sobre LPT nos dará un LLIST.

Antes de seguir adelante recomendaremos a los lectores que guarden en distintas cintas los programas grabados mediante CSAVE/CLOAD, los grabados con SAVE/LOAD y los que utilizan BSAVE/BLOAD, pues puede ser que intentemos cargar con BSAVE un programa cargado con CSAVE o SAVE, pues dada la disparidad de códigos que emplea cada instrucción para guardar los programas en cassette si se leyera un archivo de forma equívoca daría lugar a graves errores dentro de un proceso de aplicación.

Grabación de datos en cinta

Hasta ahora hemos visto cómo se guarda en cinta los programas que de hecho son archivos con un contenido muy especial, pero para poder grabar y leer datos resultantes de un programa, es decir, para organizar un archivo de datos o fichero, es necesario un poco más de preparación.

En primer lugar, no trabajaremos como para grabar un pro-

grama en modo directo sino en modo diferido, esto es dentro de un programa previo a la grabación de datos. Sería inútil tener encendido el cassette gastando cinta y también tener abierto el comando de datos del ordenador al cassette en caso de no tener remote.

Es evidente por otra parte que no siempre grabaremos todos los archivos. En ocasiones podemos hacerlo pero lo más frecuente es grabar los datos según los obtengamos a medida que avanza el proceso. Por tanto, ya podemos imaginarnos que la grabación de datos en un fichero es más trabajosa e implica un mayor número de instrucciones que la grabación de un programa. Por ello es necesario a todos los niveles de programación, que se hayan dispuesto instrucciones especiales que simplifiquen al máximo el trabajo.

La primera instrucción que vamos a estudiar es OPEN, que prepara la apertura de un fichero y cuya sintaxis es:

OPEN"(nombre dispositivo): (nombre de archivo)":

FOR	OUTPUT INPUT	A$# (número de archivo)

Sabemos que los dispositivos posibles son 4 y que a cualquiera de ellos le podemos asignar la apertura de un archivo.

De esta manera podemos abrir uno o más archivos de cassette, impresora, pantalla y pantalla de textos. Lo habitual es asignar la apertura de como máximo un archivo por cada dispositivo.

El nombre del archivo es optativo, pero recomendamos poner siempre un nombre, pues sino, es muy difícil operar posteriormente con él. Si optamos por ponerle nombre, éste, como siempre, no debe tener más de 6 caracteres pues el sistema ignora a los demás. El número de canal debe ser un entero entre 0 y 15, que es el mayor número posible de archivos con que puede trabajar el MSX. Si se abre más de un archivo es necesario dar números diferentes a cada uno de ellos. Pero esto nos obliga a utilizar previamente una sentencia MAXFILES = (n) en la que como hemos dicho N puede ser un número de 0 a 15 o cualquier variable numérica que contenga un valor entre 0 y 15. Esta sentencia es necesaria en cuanto se abre más de un archivo simultaneamente (en realidad no lo hace simultánemente

sino uno tras otro). Sin embargo, hay dos puntos muy necesarios a tener en cuenta cuando se utiliza esta sentencia.

En primer lugar, cada archivo preparado para abrirse en un programa ocupa un área de memoria de la parte de usuario, por lo que una apertura innecesaria o mal planificada de ficheros puede reducir considerablemente la memoria de que disponemos para introducir nuestros programas.

En segundo lugar, esta instrucción siempre debe aparecer en primer lugar en un programa. Más aún, en las primeras líneas tras los REM de títulos y propiedad, pues tras su ejecución se remodela toda el área de usuario (por el motivo antes expuesto) y borra todo lo que hubiéramos almacenado en estas direcciones con anterioridad, especialmente las matrices que hayamos creado con DIM, o los DEFINT, e incluso los valores que se hayan asignado a las variables, antes de la instrucción MAXFILES. Por otra parte con MAXFILES=0 sólo prodremos ejecutar CSAVE, CLOAD, SAVE y LOAD pues el canal 0 está reservado para ellas.

Así pues, tras una sentencia MAXFILES = (n) deberemos ejecutar una sentencia OPEN; si la instrucción OPEN utiliza el dispositivo cassette, tras ejecutarse y si disponemos de control de «remote», veremos como el motor del magnetófono se pone en marcha y en la cinta se graba el nombre del fichero si es un OPEN FOR OUTPUT. Si el OPEN es FOR INPUT –todavía con dispositivo cassette– cuando se ejecuta esta instrucción el cassette se pone en marcha y lee la cinta hasta encontrar el nombre de fichero que se le introdujo en el OPEN. Ejemplo de lo dicho hasta ahora:

```
MAXFILES = 2
OPEN "CAS" FOR INPUT
OPEN "CAS" FOR OUTPUT
```

Desde luego antes de empezar ninguna grabación o lectura es necesario controlar que la cinta esté en lugar correcto. De no hacerlo es muy fácil que al grabar borremos lo anteriormente grabado. Leer una buena colocación de la cinta nos evitará muchos ratos de tediosa espera pues los cassettes de audio suelen ser muy lentos.

```
10 MAXFILES=2
20 OPEN"CAS:DATA"FOR INPUT AS #2
30 INPUT #2,X%
40 PRINT"X% ES":X%
50 CLOSE
```

Ahora ya sabemos decirle al ordenador el número de canales que debe reservar para entradas/salidas de ficheros (MAXFILES) y también, asignar dispositivos y canales a ficheros, así como especificar si son de entrada o de salida (FOR INPUT; FOR OUTPUT). Todo esto lo hacemos mediante la instrucción OPEN pero para decirle que grabe o lea un dato concretamente ¡ahora! es necesario utilizar aún más instrucciones:

INPUT #, (N canal), expresión para leer datos

PRINT #, (N canal), expresión para grabar datos

El número de canal naturalmente será el especificado en la sentencia OPEN correspondiente y la expresión será lo que grabaremos o leeremos al ejecutar la orden, es decir el dato.

Para no tener que escribir tantas instrucciones INPUT # y PRINT # como datos haya que grabar, es necesario organizar algún tipo de bucle o contador. A continuación dos ejemplos de grabación de un archivo.

```
10 SCREEN,,,2
20 CLS:PRINT"Este programa graba datos e
n cinta,ponga una cinta en el magnetofon
"
30 PRINT:PRINT"pulse la tecla -S- para f
inalizar el proceso"
40 PRINT:PRINT"pulse REC+PLAY y a contin
uacion la barra de espacios"
50 IF INKEY$ <>" " THEN 50
60 PRINT "ESPERE"
70 OPEN "CAS:DATO" FOR OUTPUT AS #1
80 H%=H%+1
90 PRINT "DATO";H%;" ":INPUT HH$
95 PRINT #1,HH$
100 IF HH$<>"S" THEN 80
110 CLOSE
120 PRINT "PLSE STOP (EN EL CASSETTE)"
130 SCREEN,,,1:END
```

```
10 DIM A$(1,2)
20 OPEN "CAS:DATA" FOR OUTPUT AS #1
30 FOR A=0 TO 1
40 FOR B=0 TO 2
50 READ A$(A,B)
60 PRINT #1,A$(A,B);",";
70 NEXT B
80 NEXT A
90 CLOSE #1
100 END
110 DATA Jose,Antonio,Javier
120 DATA Pepe,Toni,Javi
```

Como puede observarse hemos introducido una nueva sentencia CLOSE cuya sintaxis es: CLOSE (#) (número de canal) (número de canal) (número de canal). El número de canal es el número de archivo que se cierra, (por omisión con sólo CLOSE se cierran todos los archivos). Con esta sentencia tras cerrar un archivo, el canal por el que recibía o enviaba datos queda libre y podemos utilizarlo para otro archivo, e incluso asignarle otro dispositivo, lo cual resulta muy interesante, pues de esta forma no necesitamos reservar espacio para estos nuevos archivos mediante MAXFILES por lo que ahorramos memoria de usuario.

Ejemplo de cierre de un archivo y apertura posterior de otro con el mismo numero.

```
10 OPEN"CAS:PARES" FOR OUTPUT AS #1
20 CLS:FOR N%=0 TO 50 STEP 2
30 PRINT#1,N%
40 PRINT N%;" ";
50 NEXT
60 CLOSE #1
70 OPEN"CAS:IMPARES" FOR OUTPUT AS #1
80 CLS:FOR H%=0 TO 50 STEP 3
90 PRINT#1,H%
100 PRINT H%;" ";
110 NEXT
```

Si cambiamos el programa anterior para introducir un mayor número de datos y lo ejecutamos, podremos observar que

aunque aparentemente graba los datos uno a uno, el motor del cassette se acciona sólo a intervalos regulares, y luego se desconecta. Esto es debido a que en la memoria que se reserva el sistema para cada canal, con la sentencia MAXFILES, se guardan los datos, hasta que la memoria reservada al canal de salida se llena, en cuyo caso graba los datos seguidos y espera que vuelva a llenarse de nuevo, si eso no ocurre, transmitirá todos los datos restantes que tenga en la memoria reservada cuando el archivo se cierre mediante la sentencia CLOSE.

Otra peculiaridad interna del sistema, que nosotros no podemos observar, es que cada vez que sacamos datos a un archivo con la sentencia PRINT # el sistema escribe automáticamente despues de cada dato, un código de retorno y un código de avance de línea (&H0D) decimal 13 (&H0A) decimal 10, estos dos códigos, nos indican la separación entre datos (véase tabla 3 código ASCII).

Cuando los datos son alfanuméricos (cadenas) tenemos la opción de sacar varios datos con una sola sentencia PRINT#, insertando "," entre cada uno de los datos, PRINT#1,A$;",";B$, la coma indica la separación entre los datos A$ y B$ que se manejan como dos datos diferentes cuando se introducen desde el archivo. Esto es especialmente útil para grabar datos asociados de dos tipos diferentes como por ejemplo Nombre N$ y Teléfonos T$:

PRINT#1 N$; ","; T$

Aquí grabará primero el nombre y luego el teléfono asociado.

Cuando los datos son de tipo numérico y se utiliza a INPUT# para leerlos, cada uno de los datos se separa automáticamente con sólo encontrar un ";" entre ellos.

Un caso muy frecuente cuando trabajamos con ficheros es encontrarnos con que no conocemos cuántos registros (series de datos) tenemos archivados. En consecuencia no podemos utilizar un bucle FOR/NEXT por lo que debemos emplear la función EOF con la que identificamos el final del archivo.

EOF (End of file) o sea en inglés «fin de archivo» es una función del BASIC que, cuando es interrogada de la siguiente forma PRINT EOF (número de archivo) nos ofrece un valor 0 si no se han acabado los datos o de –1 si no hay más datos a leer. Ello es debido a que el sistema graba una marca (&H1A) al final

de cada archivo. Si utilizamos esta función es necesario estar seguros de que el dispositivo por el que preguntamos ha sido abierto de entrada mediante una sentencia OPEN INPUT o de lo contrario los resultados pueden ser catastróficos. Quizá la mejor manera de utilizar la función EOF sea mediante una pregunta del siguiente tipo:

IF EOF (núm. archivo) = 0 THEN 100 ELSE GOTO 200

En la línea 100 insertaremos la rutina para tratamiento de registros y en la línea 200 tendremos la rutina de fin de lectura.

```
10 CLS:PRINT TAB(12)"BUSCALO":PRINT
20 PRINT:PRINT"Este programa busca datos
 por la":PRINT"inicial en el fichero dat
os. "
30 PRINT: PRINT"Pulse PLAY en su cassett
e y a continuacion la barra de espacios"

40 IF INKEY$<>" "THEN 40
50 INPUT"Escriba la inicial del nombre:
";Q$
60 OPEN"CAS:NOMBRE"FOR INPUT AS #1
70 INPUT#1,N$
80 IF EOF(1)=0 AND Q$<>LEFT$(N$,1) THEN
70
90 PRINT:PRINT"El nombre es ":N$
100 CLOSE:PRINT"Pulse STOP en el magneto
fono."
```

Otra de las aplicaciones inmediatas de testión de dispositivos de archivo del MSX es la posibilidad de escribir textos en pantallas gráficas –SCREEN 2 y SCREEN 3–, dado que en ellas es posible emplear la orden PRINT #1, según se puede apreciar en el siguiente programita:

```
10 SCREEN 2
20 OPEN "GRP:" FOR OUT PUT AS#1
30 PRESET (100,50)
40 PRINT #1, "Nombre de archivo"
50 GOTO 50
```

En la línea 10 determinamos la pantalla, mientras que en la 20 abrimos el archivo en pantalla. La instrucción de la línea 30 es de posicionado del texto en pantalla mediante PRESET y sus correspondientes coordenadas. Los caracteres a ver en pantalla de los datos de archivo se especifican en la línea 40. Si hubiésemos dibujado entre esta línea y la 20 también aparecería junto con el texto en SCREEN 2.

Como organizar sus programas de gestión desde cassette

Toda aplicación de gestión, por modesta que sea, abarcará una serie de programas interrelacionados (programas para grabar datos, para modificarlos, leerlos y realizar con ellos funciones, etc. etc.). Esta serie de programas recibe el nombre de aplicación y su utilización, según unas normas predefinidas (1.° ejecutar el programa de lectura, 2.° modificar los datos con el 2.° programa, etc. etc) se denomina proceso de datos. Analicemos esto más a fondo.

Proceso de datos

Datos son los elementos usados como base de decisión, cálculo o medida en un proceso. Los datos por sí mismos no dicen nada por lo que deben ser procesados y convertidos en información. Este proceso consiste en la selección, tratamiento y combinación de los datos, con objeto de obtener un mensaje significativo para alguien.

A tal efecto el sistema MSX dispone de varias posibilidades profesionales que le permiten crear ficheros en discos (puede controlar de forma sencilla dos controladores de disco) de varios formatos, asimismo, puede almacenar datos en cartuchos ROM de memoria viva, intercambiar datos con otros aparatos por medio de su salida RS232 y grabar datos en un formato ASCII puro, que puede hacer que intercambie datos con aparatos de la norma IBM PC (el hermano mayor de MSX por parte de Microsoft) y un sinfín de posibilidades más que sólo tienen por techo el **megabyte** de capacidad máxima que se obtiene mediante un complejo sistema de paginación, todo lo cual es materia para muchos más libros, y por lo tanto no será estudiado en éste.

En los anteriores capítulos hemos dado suficientes elementos, para organizar pequeños programas de gestión familiar, con la sola ayuda de un modesto grabador de cassettes, conectado a su MSX. Ahora bien, esta modestia en los medios utilizados, nos obliga de alguna manera a plantearnos algunas normas de trabajo, que serán imprescindibles si queremos que nuestro proceso de datos familiar funcione a la perfección.

1 - Atomice su análisis al máximo. Es decir, tras un exhaustivo análisis del problema a resolver, redúzcalo a una serie de sencillos y cortos programas y no busque crear grandes y mastodónticos programas, que difícilmente podrá controlar.

2 - Tenga muy presente que las funciones básicas de todo proceso de datos (tratamiento de los datos tendente a lograr una información) son entrada, validación, ordenación, operación, salida, y que estas fases de operación han de ser reflejadas claramente en su o sus programas.

3 - Recuerde siempre, que estos han de ser claros. No escatime impresiones por pantalla de las instrucciones necesarias para una correcta ejecución. Lo cual, se convierte en imprescindible si nuestra grabadora de cassette no dispone de control REMOTE (será necesario entonces constantes mensajes del tipo, «apriete intro y a continuación Play en el cassette» etc. etc.)

4 - No disponiendo de ficheros dinámicos, es necesario explotar al máximo las posibilidades del proceso PADRE-HIJO que consiste en leer un fichero de datos, o parte de él y manipularlo (anulando o modificando según parámetros previos), para crear de salida otro nuevo archivo-hijo, que sea reflejo del padre, que leímos previamente, pero ya modificado o actualizado.

5 - Recuerde que los tratamientos de datos alfanuméricos, carecerán de secretos para usted si domina las operaciones de cadenas.

6 - Mediante estas operaciones de cadenas, conseguirá crear en sus ficheros registros (series de datos interrelacionadas dentro de un fichero) de longitud fija o variable. En el primer caso, una función LEFT$ o similar nos puede servir para seleccionar los campos de datos

que necesitemos modificar. en el sengundo caso, se hará imprescindible crear caracteres de fin de registro (algo similar a lo que hace el sistema con EOF).

7 - Para evitar problemas, utilice con profusión el tratamiento de errores y cree sus propios códigos de errores para aquellos casos más frecuentes que halle.

8 - Dada la imposibilidad de rebobinar y correr la cinta de forma inteligente, la interrupción por intervalo puede ser útil, combinada con una orden motor, para posicionar el cassette en una vuelta concreta, a partir de la cual leer.

9 - Para evitar utilizar las sugerencias esbozadas en el punto 7, o tener que cambiar de cinta varias veces, para grabar y leer datos, según el proceso Padre-Hijo comentado en el punto 3, será mejor crear varios archivos de pequeño tamaño y en cintas diferentes que mastodónticos archivos, más sensibles a los errores y posteriores complicaciones. HAGA COPIAS DE TODOS SUS ARCHIVOS.

10 - Conscientes del ingenio necesario para organizar con sus reducidos elementos programas ambiciosos, aconsejamos al lector que no dude en acometer la tarea, pues de esta manera deberá utilizar un auténtico concepto de análisis del problema y las posibilidades, lo que en definitiva es la quintaesencia de la programación.

CAPITULO VII

Tratamiento de errores

A poco que haya trabajado con su ordenador habrá observado cómo aparecen en pantalla los mensajes correspondientes a los errores generados en el sistema. Estos errores son debidos a instrucciones mal empleadas, parámetros incorrectos o incompletos, cálculos imposibles (división por cero), operaciones ilegales con cadenas, etc. etc. Estos errores provocan en el sistema una interrupción en la ejecución del programa, la aparición en pantalla del tipo y número de error y acto seguido, pasamos a modo directo.

Los errores pueden producirse asimismo, en modo directo, obteniendo siempre el mismo tipo de mensaje, su formato es

(MENSAJE DE ERROR) IN (N.° de línea)

El número de línea corresponderá a la línea del programa que se ejecutaba en el momento de producirse el error. Pero no forzosamente, la línea en que se encuentra el error. Es decir que si en una línea 20 asignamos un valor 240 a una variable X, ello es perfectamente legal, pero si en una línea posterior, por ejemplo la 50, especificamos la instrucción STRIG (X), al no aceptar esta instrucción el valor 240, nos dará error en la línea 50, que es correcta en si misma.

Los mensajes de error que utiliza el sistema MSX van numerados del 1 al 69. Los errores del 26 al 50, inclusives, son reservados para futuras expansiones del sistema, y del 51 hacia adelante están dirigidos a la gestión de archivos, de los cuales el bocado del león se lo llevan los dispositivos de acceso directo, por lo que tienen efecto si la unidad básica no utiliza sistema de diskettes.

Por otra parte el sistema reserva los códigos de error del 70 al 255 para utilización del usuario, de la forma que veremos enseguida.

El conocimiento de estos mensajes y sus códigos, nos será muy útil para prevenir bloqueos en nuestros programas. Bloqueos que pueden ser evitados mediante el potente juego de instrucciones MSX BASIC para el tratamiento de errores.

El tratamiento de errores esta muy relacionado con el sistema de interrupciones de que dispone el sistema, que tratamos en otro capítulo. La primera sentencia por tanto, nos resulta familiar: ON ERROR GOTO (numlinea).

Esta instrucción produce en caso de detección de error un GOTO a la línea especificada, en la sentencia, con inhibición del mensaje en la pantalla. En este momento el sistema envía para su posterior tratamiento, el número de línea en que se produce el error y el código del error a las variables reservadas por el sistema ERL y ERR respectivamente.

De esta manera podemos seleccionar según el número de error almacenado en ERR algunos errores determinados, que pueden solventarse mediante una subrutina especial o, simplemente, traducir los textos, y hacer que aparezcan en castellano. En este último caso, no será necesario continuar adelante, pero si hemos solventado el problema causante del error en una rutina de interrupción por error, podemos volver a la ejecución del programa mediante la sentencia:

RESUME (numlinea)

En esta sentencia, si omitimos el número de línea o lo hacemos 0, el sistema devuelve el flujo del programa y va a la línea causante del error otra vez, e intenta ejecutarla. En cambio si in-

Tabla mensajes de error

1 NEXT without FOR	: No hay sentencia FOR correspondiente a su sentencia NEXT.
2 Syntax error	: Error sintáctico en la sentencia.
3 RETURN without GOSUB	: No hay sentencia GOSUB correspondiente a su sentencia RETURN.
4 Out of DATA	: No ha más datos para leer.
5 Illegal function call	: Especificación ilegal en una función o un mandato.
6 Overflow	: Demasiados o demasiado pocos datos.
7 Out of memory	: No hay más memoria.
8 Undefined line number	: Se ha especificado un número de línea no definido.
9 Subscript out of range	: Subíndice de matriz fuera del margen definido.
10 Redimensioned array	: La matriz de una sentencia DIM ya ha sido especificada.
11 Division by zero	: Se ha dividido por cero.
12 Illegal direct	: El mandato no puede utilizarse en modo directo.
13 Type mismatch	: Error de deletreo de los datos.
14 Out of string space	: No queda más área para variables alfanuméricas.
15 String too long	: Cadena demasiado larga.
16 String formula too complex	: Cadena demasiado complicada.
17 Can't CONTINUE	: No es posible continuar la ejecución del programa.
18 Undefined user function	: Se ha utilizado una función no definida mediante una sentencia DEF FN.
19 Device I/O error	: Error en el equipo conectado.
20 Verify error	: Discordancia entre el programa del casete y el de la memoria.
21 No RESUME	: No hay sentencia RESUME correspondiente a su sentencia ON ERROR.
22 RESUME without error	: No hay sentencia ON ERROR correspondiente a su sentencia RESUME.
23 Unprintable error	: Ha ocurrido un error carente de mensaje de error.
24 Missing operand	: Falta un operando.
25 Line buffer overflow	: El programa introducido sobrepasa el tamaño de la memoria intermedia.
51 Internal error	: El contenido de la memoria o el texto es anormal.
52 Bad file number	: Nombre de archivo incorrecto.
54 File already open	: El archivo especificado ya está abierto.
55 Input past end	: Ya se han leído los últimos datos.
56 Bad file name	: Especificación incorrecta de archivo.
57 Direct statement in file	: Durante la carga de un archivo se ha introducido un mandato en modo directo.
59 Finè not OPEN	: Hay que abrir el archivo.

troducimos, tras RESUME un número de línea, la ejecución, continúa en esta línea. Con la opción

RESUME NEXT

la ejecución se desplaza a la línea siguiente a la que produjo el error.

```
10 ON ERROR GOTO 60
20 INPUT A
30 B=SQR(A)
40 PRINT "sqr(a)=";B
50 END
60 IF ERR=5 AND ERL=30THEN70
70 PRINT "introduce un numero positivo"
80 RESUME 20
```

Otra posibilidad es la de definir nuestros propios errores mediante la instrucción ERROR (num error).

Si hemos preparado previamente el programa mediante una instrucción ON ERROR, cuando encuentre la sentencia ERROR 71 se creará una interrupción por error, acumulándose en ERR el número de error que le hemos dado, (en este caso 71), y en ERL el número de línea en que encontró la sentencia ERROR 71, a continuación, ejecutará la rutina de error a la que le envía la sentencia ON ERROR GOTO. En ella, podemos preguntar por el código 71, pues ya ha sido definido previamente.

Estamos pues ante un caso de error provocado por nosotros, es decir basta decirle a la máquina ERROR 24, para provocar un error del tipo MISSING OPERAND (obtendremos el mensaje y la interrupción). Lo mismo ocurre con todos los números de error del 1 al 69, que son reservados para el BASIC, es decir, forzamos el error correspondiente al número introducido en el parámetro de ERROR. Para ello no es necesario que haya una declaración previa de ON ERROR. Si diéramos al parámetro un valor mayor de 69 el mensaje que aparece es «UNPRINTABLE ERROR», pues este error no lo tiene definido el sistema, y es necesario tratarlo siempre mediante una subrutina de error generada por ON ERROR.

La forma más corriente de utilizar ERROR (numerror) es mediante una disyuntiva, del tipo IF (condición) THEN ERROR (numerror) ELSE (etc) o bien IF (condición) THEN (sentencia) ELSE ERROR (n.º).

Es decir que si una determinada condición se cumple o no se considera error con lo cual generamos una interrupción y la enviamos a rutina general de tratamiento de errores, que puede servir para todos los que consideremos que pueden producirse en la ejecución de un programa. En esta rutina preguntamos por el valor de ERR y si es el (n.º) número introducido en la sentencia ERROR lo tratamos según nos interese. Si por el contrario, se produce una interrupción por error detectado por el sistema, en la misma rutina anterior, podemos darle a éste un tratamiento diferente.

Existe, la posibilidad de que se produzca un error dentro de una rutina de interrupción por error, en cuyo caso el error es tratado normalmente por el sistema mediante el correspondiente mensaje y el bloqueo en la ejecución. No hay, por tanto, riesgo de entrar en un bucle infinito, que se formaría, si este segundo error nos remitiera de nuevo esa misma rutina causante del nuevo error.

Con una sentencia del tipo ON ERROR GOTO suprimimos el efecto de ON ERROR (numlínea). Si aparece durante la ejecución de una rutina de error. En este caso el programa pasa a modo directo y aparece el correspondiente mensaje de error en la pantalla. La omisión del número de línea, por el contrario, sólo produce un error del tipo «Undefined line number».

Los ejemplos que damos a continuación clarificarán más lo visto hasta aquí.

```
10 PRINT "Teclear Control-STOP para sali
r"
20 ON ERROR GOTO 50
30 LIST
40 GOTO 70
50 IF ERR=19 THEN PRINT "Listado detenid
o"
60 PRINT"En Linea";ERL;:ERROR ERR
70 ON ERROR GOTO 0' no hay desvio, regre
so a la normalidad
80 PRINT "SE ACABO"
```

```
10 CLS
20 ON ERROR GOTO 80
30 PRINT:INPUT "INTRODUCIR UN NOMBRE ";N
OM$
40 PRINT:PRINT NOM$
50 PRINT:INPUT "SE CONTINUA (SI/NO)";A$
60 IF A$<>"SI" AND A$<> "NO"THEN ERROR 2
00
70 IF A$="SI" THEN GOTO 30 ELSE END
80 REM RUTINA DE DESVIO
90 IF ERR=200 THEN PRINT:PRINT"INTRODUCI
R SI O NO":RESUME 50
100 PRINT:PRINT "ERROR DESCONOCIDO"
```

```
10 CLS
20 ON ERROR GOTO 70
30 INPUT "INTRODUCIR UN NUMERO DISTINTO
DE 0 ";N
40 X=1/N
50 PRINT:PRINT "INVERSO DE ";N;"=";X
60 PRINT:GOTO 30
70 REM RUTINA DE TRATAMIENTO DE ERRORES
80 IF ERR<>11 THEN END
90 PRINT :PRINT "HAY QUE INTRODUCIR UN N
UMERO DISTINTO DE 0 "
100 PRINT
110 PRINT "Codigo de error:";ERR
120 PRINT
130 PRINT "Linea de error:";ERL
140 PRINT
150 RESUME 30
```

CAPITULO VIII

Los gráficos en MSX

Una de las características en que más hincapié hacen los fabricantes de este estándar es su gran versatilidad gráfica. Todos los MSX van equipados con un procesador de vídeo (VDP) de Texas Instruments –TMS-9924A–, que dispone de una capacidad propia de 16 K y funciona básicamente en cuatro modos, que en BASIC corresponden a SCREEN 0, 1, 2 y 3. Los modos 0 y 1 son de texto, mientras que el 2 y 3 son gráficos y por los cuales podemos acceder a todos los puntos de la pantalla mediante instrucciones especiales.

En Screen 2 hay 256 pixels (puntos) por 192 líneas a los que se puede acceder individualmente. SCREEN 3 es de 256×192 puntos, pero están agrupados en bloques de 4×4 es decir, 64×48 bloques de puntos. La selección de modo gráfico tan sólo es posible en modo diferido. Si se trabaja en modo directo, tras un breve destello de la pantalla, ésta se borra y vuelve al modo de texto.

La instrucción COLOR en modo gráfico no modifica instantáneamente el contenido de la pantalla, sino el color de escritura para las instrucciones gráficas. Por lo tanto, parece posible seleccionar en modo gráfico el color para cada uno de los puntos de la pantalla, pero en realidad esto sólo podemos hacerlo en baja resolución ya que en SCREEN 3 cada pixel es un número de color de 0 a 15.

Es posible mezclar texto y gráficos abriendo el archivo de pantalla mediante la sentencia OPEN «GRP»: AS# (numfichero) y posteriormente con PRINT# (numfichero), «texto» escribir el texto deseado.

Además de las instrucciones gráficas (PSET, LINE, CIRCLE...), los ordenadores MSX poseen una potentísima instrucción que utiliza el GML (Macro Lenguaje Gráfico). Nos referimos a la instrucción DRAW que reune características de todas las demás instrucciones gráficas y posibilita asimismo dibujar en coordenadas absolutas. Es decir, que dando valores numéricos fijos o relativos (cuando se trate de variables), permite ajustar la escala de una figura controlando sus dimensiones y someterla a rotaciones diversas de izquierda a derecha y de arriba abajo.

Con MSX también es posible la creación de SPRITES, es decir, definir caracteres propios dándoles forma y dimensiones para poderlos introducir dentro de los gráficos y crear animación.

Todos los comandos gráficos se refieren a una posición del «cursor gráfico» en la pantalla. La coordenada horizontal (abscisa) siempre debe estar comprendida entre 0 y 255, mientras que la coordenada vertical (ordenada) debe estarlo entre 0 y 191. El punto 0,0 (punto de origen) esta situado en la esquina superior izquierda de la pantalla.

Por lo tanto, la esquina superior derecha es el punto 255,0; la inferior izquierda el 0,191 y la inferior derecha el 255,191.

0,0	255,0
0,191	255,191

Hemos comentado anteriormente que cuando se seleccione un modo gráfico debe hacerse un programa, puesto que en el caso de que se haga en modo directo simplemente se producirá un destello en pantalla y se regresará al modo de texto. Para evitar esto, le sugerimos cuatro opciones:

1.–
```
10 SCREEN 2 (o 3)
–PROGRAMA–
1000 GOTO 1000
```

2.–
```
10 SCREEN 2 (o 3)
–PROGRAMA–
1000 z$ =INPUT$ (1)
```

3.–
```
10 SCREEN 2 (o 3)
–PROGRAMA–
1000 IF INKEY$=
""THEN 1000
```

4.–
```
10 SCREEN 2 (o 3)
–PROGRAMA–
1000 IFINKEY$<>"X"
THEN 1000
```

La primera de ellas es la llamada «bucle infinito». Para detener el programa y volver a modo de texto debemos pulsar conjuntamente CTRL y STOP. La segunda y la tercera son prácticamente la misma. En el momento en que pulsemos cualquier tecla, se interrumpirá el programa y volveremos al modo de texto, no exactamente en el acto sino cuando haya finalizado la ejecución. Esto se debe a que la máquina no realiza la inspección del teclado hasta que ha realizado el programa. La cuarta opción no detiene el programa a no ser que se pulse una tecla específica (en el caso del ejemplo la «x»).

Cuando trabajemos en SCREEN 3 debemos tener en cuenta que las instrucciones gráficas que utilicemos efectúan una aproximación sobre las coordenadas dadas. Para saber cuales son las correspondencias entre los puntos de SCREEN 2 y SCREEN 3 a la hora de efectuar correcciones entre uno y otro modo, debemos tener en cuenta lo siguiente:

Todos los puntos cuyas abscisas estén comprendidas entre $0 + X$ y $3 + X$ en SCREEN 2 corresponden a la abscisa $0 + X$ en SCREEN 3.

Todos los puntos cuyas ordenadas estén comprendidas entre $0+Y$ y $3+Y$ en SCREEN 2 corresponden a la ordenada $0+Y$ en SCREEN 3.

Por lo tanto, X debe ser un entero comprendido entre 0 y 250 divisible por 4 e Y también debe ser un entero, comprendido entre 0 y 188; divisible por 4.

Instrucciones gráficas

PSET/PRESET
SINTAXIS: PSET (o PRESET) (STEP) (X,Y) (COLOR)

Sirven respectivamente (si no se especifica COLOR) para situar o borrar un punto en las coordenadas X, Y de la pantalla. Si no se especifica el valor de STEP el punto se situará en las coordenadas absolutas; si se especifican las coordenadas mediante variables, y se les asigna un valor STEP, X e Y servirán para definir un desplazamiento sobre las coordenadas.

Las especificaciones de COLOR, PSET y PRESET cumplen la misma función es decir, ambas dibujan un punto, situándose el cursor en las coordenadas del punto.

EJEMPLOS:

```
10 SCREEN 3: FOR A=1 TO 100
20 PSET (A,A)
30 PRESET (A,A): NEXT
40 Z$=INPUT$(1)
```

```
10 SCREEN 3
20 FOR A=1 TO 100 STEP4
30 PSET STEP (A,A), 15
40 PRESET (A,A), 1: NEXT
50 Z$=INPUT$(1)
```

POINT
SINTAXIS: POINT (X,Y)

La función POINT nos da el color del punto en las coordenadas X, Y. Si las coordenadas quedan fuera del área de visualización obtendremos –1.

LINE
SINTAXIS: LINE [(STEP) (X1,Y1)]–(STEP) (X2,Y2), (COLOR), (B o BF)

LINE es una de las sentencias más utilizadas para dibujar. Tiene una gran versatilidad ya que puede trazar líneas, hacer figuras llenas y perfiles. Si no se especifica un color, se toma el color por defecto. La opción B dibuja un contorno cuadrado y BF un cuadrado lleno. Las coordenadas del punto pueden ser omitidas (no el guión) en cuyo caso se traza una línea entre la posición del último punto definido y el segundo punto. Si utiliza-

mos coordenadas relativas e incrementos mediante STEP podemos (entre otras cosas) dibujar haciendo referencia a coordenadas absolutas. Esto permite presentar un dibujo en diferentes posiciones o bien, a distintas escalas. Es posible omitir una o varias opciones, pero se deben sustituir las opciones no utilizadas por comas en el caso de que se desee utilizar una opción posterior.

```
1 ' LINE
10 SCREEN 2:DEFINT A-Z:COLOR1,15,11
20 FOR A= 1 TO 255STEP19:FORB=1 TO 191STEP 11
30 FOR C= A/2TOBSTEP7:FOR D=B*2 TO 1STEP 38
40  LINESTEP(A,C)-(D,B),13
50  LINESTEP(D,C)-(B,A),12
60  LINESTEP(C,A)-(B,D),11
70  LINE STEP(A,C)-(B,D),10
80 NEXT
90 NEXT
100 NEXT
110 NEXT
120 Z$=INPUT$(1)
```

PAINT
SINTAXIS: PAINT (STEP) (X,Y), (COLOR), (COLOR del borde)

PAINT rellena una zona de la pantalla o una figura con el color elegido o por defecto en caso de no emplear la opción COLOR. El color del borde se utiliza en SCREEN3 si se desea bordear el dibujo, pero en SCREEN2 no se usa ya que automáticamente toma el mismo valor que COLOR y por lo tanto puede suprimirse. Las coordenadas (X,Y) deben estar dentro de la figura y/o de la pantalla, no como en otros comandos gráficos. Tan sólo actúa en sentido horizontal y vertical, y si no se observa escrupulosamente que la figura a rellenar este absolutamente cerrada nos encontramos con la desagradable sorpresa de que PAINT huye por la abertura y nos borra la pantalla. STEP sirve para definir un desplazamiento por el eje de coordenadas relativo a la posición actual del «cursor gráfico». De todos modos,

no es quizá la mejor opción para rellenar un área. Con mucho, la opción BF es más rápida aunque, bien es verdad que es más complicada de utilizar.

```
1  'PAINT
10 SCREEN 2:COLOR 1,14,12
20 LINE(40,40)-(190,100)
30 LINE(60,170)-(190,100)
40 LINE(40,40)-(60,170)
50 PAINT (100,100),1
100 X$=INPUT$(1)
```

CIRCLE
SINTAXIS: CIRCLE (STEP) (X,Y), RADIO (COLOR) (ANGULO INICIO) (ANGULO FIN), ASPECTO

CIRCLE representa el único medio de obtener en pantalla formas redondeadas sin utilizar funciones trigonométricas, que son por otro lado más lentas.

Traza un círculo, arco o elipse cuyo centro se sitúa en las coordenadas absolutas X,Y o bien, caso de especificar STEP las coordenadas sirven para definir un desplazamiento sobre el eje relativo a la posición actual del «cursor gráfico». Los ángulos inicial y final pueden omitirse. Caso de especificarse, permiten dibujar un segmento de la circunferencia o elipse.

Los ángulos son tomados en radianes y representan valores entre 0 y 2*PIradianes. El aspecto sirve para definir la relación entre los radios horizontal y vertical de una elipse. La relación se divide por la altura para obtener la anchura, convirtiéndose en radio la menor de estas magnitudes. Si no se especifica, la máquina toma el valor por defecto.

```
10 CLS:PRINT"DIBUJO DE UN ARCO
20 PRINT"PULSA UNA TECLA"
30 Z$=INPUT$(1)
40 SCREEN 2:COLOR 1,14,12
50 CIRCLE(100,100),40,8,1,6,1
60 Z$=INPUT$(1)
```

```
10 SCREEN 2:COLOR 1,14,12
15 FOR X= 30TO190 STEP12
20 CIRCLE(X,100),20,8,,,1.2
25 NEXT
30 Z$=INPUT$(1)
```

```
10 SCREEN 2:COLOR 11,1,1
20 X= 128:Y=96:R=45:C=15:S=0:E=6.2:A=.4
30 PI=ATN(1)*4:R1=R
40 FORT#=1 TO360 STEP 10
50 W=2*PI*T#/360
60 X=130+R1*COS(W):Y=95+R1*SIN(W)
70 GOSUB 100
80 NEXT T#
90 GOTO 90
100 CIRCLE (X,Y),R,C,S,E,A:RETURN
```

```
10 'graficos sinusoidales
20 SCREEN2:CLS
30 B=RND(1):B=B*15+1
40 PI!=4*ATN(1)
50 A=RND(1):A=A*355
60 W=RND(1):W=W*100+10
70 A1=-B*PI!
80 A2=B*PI!
90 C=(A2-A1)/A
100 G=255/(A2-A1)
110 FOR L=A1 TO A2 STEP C
120 X=L*G:Y=W*SIN(L)
130 X1=130+X:Y1=90+Y
140 LINE (X1,Y1)-(X1,X1),B
150 NEXT L
160 GOTO 30
```

DRAW
SINTAXIS: DRAW (cadena alfanuméric



Nos encontramos no ante un comando
simo macrolenguaje gráfico (GML). El di
pende de una serie de comandos residentes
caracteres. Al ser posible trabajar con variables dentro de la cadena, podemos variar las escalas, memorizar formas y obtener efectos de **Zoom**. Cada comando esta formado por una letra habitualmente seguida por un parámetro. Podemos dividir los comandos que usa DRAW en dos grupos:

–De uso general

A(n) determina el ángulo de giro para cada dirección especificada. El valor (n) debe estar entre 0 y 3, correspondiendo 0 a la orientación habitual, 1 a un giro de 90° en sentido inverso a las manecillas del reloj, 2 a un giro de 180° y 3 a uno de 270°. Por defecto, se toma 0.

C(n) determina el color del trazo. Si no se especifica se toma el valor por defecto y afecta a todo aquello que se dibuje sin el prefijo «B».

S(n) Factor de escala. El argumento (n) puede ir de 1 a 255. Este factor se obtiene dividiendo el argumento por 4; es decir, si n=2 la escala será 2/4 y significará la división por 2 de todas las longitudes.

X(n) Este comando sirve, (al igual que en PLAY, como veremos cuando traemos del MML) *para ejecutar una «subcadena»* dentro de una cadena. Debe finalizar preceptivamente con punto y coma (;).

–De uso concreto

Son estos los comandos de movimiento cuya misión es generar un desplazamiento del «cursor gráfico» con trazado de una línea a partir de su última posición especificada. Tras la ejecución de cada orden las coordenadas del cursor se sitúan en el último punto dibujado, convirtiéndose esta posición en la nueva posición del cursor.

U(n). Genera un desplazamiento hacia arriba.
D(n). Genera un desplazamiento hacia abajo.
L(n). Genera un desplazamiento hacia la izquierda.

R(n). Genera un desplazamiento hacia la derecha.
E(n). Genera un desplazamiento en diagonal hacia arriba a la derecha (45).
F(n). Genera un desplazamiento en diagonal hacia abajo a la derecha (315).
G(n). Genera un desplazamiento en diagonal hacia abajo a la izquierda (225).
H(n). Genera un desplazamiento en diagonal hacia arriba a la izquierda (135).

También es posible desplazar el cursor a un punto concreto de la pantalla mediante el comando M(X,Y) que sirve tanto para coordenadas absolutas como relativas. Para usar estas últimas se debe preceder los argumentos de X e Y de los signos + o – en cuyo caso, el cursor se desplaza un número de puntos determinados por los valores de X e Y y no a las coordenadas de X,Y.

Asimismo existen dos prefijos que deben preceder cualquier comando de movimiento. Se trata de B(n); que desplaza el cursor sin dibujar y de N(n) que desplaza el cursor, dibuja y vuelve a la posición previa.

```
10 REM PRIMER EJEMPLO DE DRAW
20 SCREEN 2
30 PSET (100,50)
40 G$="c15s30u6r4d1l3d4r2u1l1u1r2d3l4"
50 G1$="c15s30u6r4d1l3d4r2u1l1u1r2d3l4"
60 DRAW G$
70 PSET (120,100)
80 DRAW G1$
90 Z$=INPUT$(1)
```

```
10 REM SEGUNDO EJEMPLO DE DRAW
20 REM CAMBIO DE ORIENTACION
30 SCREEN 2
40 PSET (120,100)
50 FOR W=0 TO 3
60 G$="A=W;C15S30U6R4D1L3D4R2U1L1U1R2D3L
4"
70 DRAW G$
80 NEXT
90 Z$=INPUT $(1)
```

```
10 REM TERCER EJEMPLO DE DRAW
20 REM ZOOM
30 SCREEN 2
40 PSET (120,100)
50 FOR W=10 TO 50 STEP 10
60 G$="A0C15S=W;U6R4D1L3D4R2U1L1U1R2D3L4
"
70 DRAW G$
80 NEXT
90 Z$=INPUT $ (1)
```

```
DRAW cuarto
10 'puntos cardinales
20 SCREEN 2
30 DRAW "bm120,100"
40 DRAW "nu10ne10nr10nf10nd10ng10nl10nh10"
50 Z$=INPUT$(1)
```

```
DRAW cinco
10 'puntos cardinales zoom
20 DEFINT A-Z
30 SCREEN 2
40 DRAW "bm120,100"
50 FOR Z=1 TO 40 STEP 5
60 DRAW "s=z;nu10ne10nr10nf10nd10ng10nl10nh10"
70 NEXT
80 Z$=INPUT$(1)
```

Sprites

El BASIC MSX dispone de un grupo de cuatro instrucciones destinadas a la creación y control de caracteres gráficos programables (SPRITES). Cambiando un solo byte de un SPRITE podemos modificar su color o su diseño. Se pueden visualizar hasta 32 SPRITES, distinguibles entre ellos por su color, diseño y coordenadas y además por su tamaño (normal o ampliado al doble) y su longitud (8 bytes cuando sean matrices de 8*8, 32 bytes cuando sean de 16*16). Las coordenadas y la longitud deben ser previamente definidas en SCREEN (excepto en SCREEN 0, donde no funcionan los SPRITES).

Hemos dicho que una de las ventajas de MSX es que permite usar hasta 32 SPRITES. Esto quiere decir que hay hasta 32 planos posibles de visualización, pero en cada uno de ellos sólo se puede incluir un SPRITE. El SPRITE 0 corresponde al primer plano de visualización; es decir, es la figura que tiene prioridad sobre cualquier otra que aparezca detrás suyo.

En SCREEN 1, 2 y 3 se pueden definir SPRITES mediante la variable de cadena especial SPRITE$ y desplazarlos por toda la pantalla con la orden PUT SPRITE, que borra el antiguo emplazamiento de la figura dibujándola en el nuevo emplazamiento asignado. El modo más fácil de dibujar un SPRITE es mediante líneas DATA de números binarios (p.e., DATA&B00111100). Si desarrollamos los SPRITES mediante este sistema de DATAS veremos en el listado la forma del SPRITE, y a los bits que estén en 1 (los encendidos) nos darán la forma y los que estén a 0 (los apagados) no se visualizarán. Un sistema muy útil para diseñar SPRITES consiste en dibujar en un papel una matriz de tantas filas y columnas como deseemos que tenga el SPRITE y rellenar los espacios que nos interesen para obtener la forma deseada. Así podremos codificarlo en binario de una manera muy simple:

1	2	3	4	5	6	7	8	
	X	X	X		X			=&B01110100
0	1	1	1	0	1	0	0	

De todos modos, esta manera de crear SPRITES mediante DATAS en binario no es la más adecuada. Si queremos ahorrar memoria lo más práctico resulta traducir el número binario a decimal mediante ?&B... (?&B01110100 nos dará 116). Para seleccionar el tamaño de los SPRITES utilizaremos la orden SCREEN (1, 2 o 3), (0 si son de 8×8 sin ampliar; 1 si son ampliados; 2 si son de 16×16 sin ampliar, 3 si son de 16×16 ampliados).

```
10 screen 2,0
20 S$=""
30 for c=0 to 8: read c$
40 S$=S+chr$(val("&B"+c$))
50 nextc
60 SPRITE$(0)=c$
70 end
80 data 00111100
90 data 01000010
100 data 01100110
110 data 00111100
120 data 01100110
130 data 00011000
140 data 00100100
150 data 11100111
```

Un SPRITE de 16*16 no es otra cosa que 4 SPRITES de 8*8 situados uno junto a otro formando un cuadrado y definidos de manera idéntica:

S$ (1)	S$ (3)
S$ (2)	S$ (4)

Una vez definidos cada uno de los SPRITES dentro de cadenas de caracteres, asignaremos su suma a SPRITE$ de la manera siguiente:

SPRITE$(1)=S$(1)+S$(2)+S$(3)+S$(4)

La instrucción que nos permitirá mover los SPRITES por toda la pantalla es PUT SPRITE, y su sintaxis es:

PUT SPRITE numplano(, (STEP), (X,Y))(,COLOR)
(,numSPRITE)

Para situar un SPRITE en la pantalla en las coordenadas X,Y, previamente debe estar definido mediante SPRITE$. La

presentanción de SPRITES en pantalla es independiente de su contenido. La abscisa (X) puede tener valores comprendidos entre –32 y 255 mientras que la ordenada (Y) debe estar entre –32 y 191. El número de plano representa el plano de visualización del SPRITE y debe estar comprendido entre 0 y 31. STEP sirve para trabajar en coordenadas relativas. Si es omitido, la esquina superior izquierda del SPRITE se sitúa en las coordenadas X, Y.

COLOR sirve para definir el color del SPRITE. Caso de ser omitido, se toma por defecto el color seleccionado para el primer plano. NUMSPRITE es el número asignado al SPRITE con SPRITE$ (n). Debemos tener en cuenta que no se puede visualizar más de cuatro SPRITES simultáneamente. Caso de intentar visualizar un quinto SPRITE desaparecería el que tuviera el último número de plano.

Si el valor de la ordenada es igual a 208, todos los SPRITES con un número de plano más alto desaparecen de la pantalla hasta que les sea asignado un nuevo valor. Si en cambio el plano asignado es 209, el SPRITE al que le es asignado desaparece. Esta posibilidad resulta muy interesante ya que facilita la animación. Otra regla importante a tener en cuenta si deseamos trabajar con animación es alternar diferentes formas rápidamente para que los caracteres parezcan estar moviéndose.

Las siguientes instrucciones tratan las bifurcaciones e interrupciones de los SPRITES y son tratadas también en el capítulo destinado a las interrupciones.

ON SPRITE GOSUB (numlinea)

Sirve para bifurcar un programa hacia un subprograma. Debe ser valida por SPRITE ON. Posibilita la desviación del programa por la colisión de dos SPRITES en la pantalla.

SPRITE ON-OFF-STOP

SPRITE ON valida el desvío para la desviación ON SPRITE GOSUB. SPRITE OFF hace que se deje de tener en cuenta ON SPRITE GOSUB. SPRITE STOP memoriza la colisión de dos SPRITES pero no valida el desvío hasta que encuentra SPRITE ON.

CAPITULO IX

El sonido del MSX

Una importante característica de los ordenadores MSX es su capacidad de generar sonidos. Actualmente esto es algo que no sorprende a nadie, puesto que la mayoría de micros disponibles en el mercado tienen esa capacidad. Lo importante en este caso no es tan sólo lo que hace MSX sino también lo fácil que lo hace.

Cualquier MSX va equipado con un «chip» de sonido. AY-3-8910 de General Instrument. Este Generador Programable de Sonido (PSG), permite mediante la instrucción PLAY el acceso al Macrolenguaje Musical (MML), de funcionamiento similar al macrolenguaje gráfico y que permite efectuar composiciones hasta a tres voces en ocho octavas. Otra opción acústica que tenemos con MSX es el acceso directo al Generador de sonido utilizando el comando SOUND. Vamos a hablar del Macrolenguaje musical.

Todas las notas de la escala –hasta 8 octavas como hemos visto– han sido incorporadas al MML. Para hacer sonar cualquier nota simplemente tiene que introducir la nota en cuestión con «notación inglesa». En notación inglesa, las notas en lugar de llamarse do, re, mi... se definen sólo mediante una letra, como veremos en el ejemplo:

DO	RE	MI	FA	SOL	LA	SI	
C	D	E	F	G	A	B	(notación inglesa)

De tal modo, cuando deseemos programar un DO pondremos una C, si es un MI una E etc. Los semitonos (sostenidos y bemoles), que son las teclas negras del piano se expresan en MML con los signos + o # para los sostenidos y – los bemoles. Hay que tener en cuenta que intervalos como MI-FA o SI-DO son tan sólo de medio tono, y por lo tanto la máquina no interpreta correctamente la nota que pretendemos que suene si introducimos por ejemplo un SI sostenido (B#) o un FA bemol (F–).

Hemos dicho antes, que la llave del Macrolenguaje musical es PLAY, cuya sintaxis es la siguiente:

PLAY (cadena alfanum) ((,cadena alfanum)(,cadena alfanum))

Las tres cadenas alfanuméricas indican que se puede componer música a una, dos o tres voces mediante esa instrucción.

Con lo que hemos visto hasta ahora, podemos atrevernos a escribir como mínimo una escala en MML. La de DO mayor –la más sencilla–, es como sigue:

PLAY «CDEFGABO5C»

Habrá notado que en medio de la secuencia hemos incluido «O5». Tiene una explicación muy simple: La distancia entre DO y el siguiente DO se denomina octava en música. Se trata de la misma nota, pero una vez es más aguda que la otra. MSX toma como valor de octava por defecto, el DO central del piano (O4). En el ejemplo anterior, como no hemos indicado ninguna octava, la máquina ha comenzado en el DO central; pero para completar la escala y acabar en DO tenemos que indicarle que el último DO es más agudo, cosa que hemos hecho al incluir «O5», es decir que hemos indicado a la máquina que toque una serie de notas comenzando en el DO de la octava cuarta y que termine con el DO de la octava quinta. Los valores de «O» deben estar comprendidos entre 1 y 8. Si volvemos a ejecutar la línea anterior, nos encontraremos con que se inicia en la octava quinta esta vez. Para evitar esto, hagamos lo siguiente:

PLAY«O4CDEFGABO5C»

Al no indicarle a la máquina en que octava debe comenzar toma como valor por defecto la octava en que finaliza. Para evitar esto, es conveniente indicar siempre en que octava deseamos comenzar.

Con el MML también es posible controlar la duración de las

notas. Por defecto, el ordenador asigna a todas las notas el valor L4 (negra), pero las longitudes de las notas y silencios, se indican al ordenador mediante L y R respectivamente, según la tabla siguiente:

	NOTAS			SILENCIOS
L1	(redonda)	4	tiempos	R1
L2	(blanca	2	tiempos	R2
L4	(negra)	1	tiempos	R4
L8	(corchea)	1/2	tiempos	R8
L16	(semicorchea)	1/4	tiempos	R16
L32	(fusa)	1/8	tiempos	R32
L64	(semifusa)	1/16	tiempos	R64

Los tresillos de corchea pueden introducirse con L9, y los múltiplos y divisores de 9 nos los darán para cualquier otra duración. En MML también se pueden indicar las notas en forma numérica mediante «N» seguida de un número entre 0 y 96, teniendo en cuenta que mediante esta notación, las notas se cuentan semitono a semitono. De tal modo, N36 es «O4C» y N35 «O4C+». Los puntos (.) en MML cumplen la misma función que los puntillos en música, es decir, aumentan la duración de la nota en la mitad de su valor.

Podemos también variar el «tempo», es decir, la velocidad de ejecución de la melodía, mediante la orden «T». El tempo por defecto en MML es «T120» –ciento veinte negras por minuto– y sus valores pueden oscilar entre 32 y 255. Si se produce algún error, o por algún otro motivo suena BEEP (chr$) (7)$ se vuelve al tempo por defecto.

PLAY«T200O4L8C.DE.FG.AB.O5C2»
(obsérvese que C2 es lo mismo que L2C)

También podemos modificar el volumen utilizando (V) seguida de un número entre 0 y 15. El volumen por omisión es V8.

En los ejemplos que hemos ido viendo hasta ahora, hemos utilizado PLAY como instrucción en modo directo para una sola voz.

Evidentemente, también puede usarse en modo diferido y almacenar su contenido en variables de cadena:

```
10 A$=«T200O4L8C.DE.FG.AB.O5C2»
20 PLAYA$
```

Podemos también definir una subrutina en MML mediante «X (nombre de variable);»:

```
10 A$=«T200O4L8C.DE.FG.AB.O5C2»
20 PLAY «O3CDEFGABXA$;»
```

Como podemos imaginar fácilmente, es posible anidar varias «subcadenas» empleando X, lo cual aumenta infinitamente las facilidades de composición.

Si deseamos extender la composición a dos o tres voces, sencillamente tenemos que definir cada una de las que vayamos a utilizar y ejecutarlas mediante PLAY separadas por una coma:

```
10 A$ = «T200O4L4CDEFGABO5CO4GEC2»
20 V$ = «T200O6L16BAGF AGFE GFED FEDC EDCO5B
O6DCO5BA O6CO5BAG AGFE GFED L2 G»
30 C$ = «T200O3 L2C.E.G.O4C.L2E»
40 PLAY A$, B$, C$
```

Existen todavía dos instrucciones que podemos utilizar con PLAY: La selección de forma de onda (Sn), cuyo argumento «n» puede valer de 0 a 15 y cuyo valor por defecto es 1 y la modulación o período (Mn), cuya «n» esta comprendida entre 0 y 65535, siendo su valor por omisión 255. De todos modos, estas «subinstrucciones» tienen íntima relación con SOUND, que pasamos a ver ahora. Digamos simplemente que si se desea modificar el sonido de PLAY (su timbre), debe hacerse utilizando S y M. Por ejemplo:

```
PLAY «S1M5000CDEFFEDC»
```

Nos interpreta esas notas en «staccatto».

Una cosa más:

Si utilizamos «S», perdemos todo el control sobre el volumen (V).

Aún hay otra utilización de PLAY:

```
a = PLAY (número voz)
```

Nos indica si hay o no música interpretándose. Número voz debe estar comprendido entre 0 y 3 siendo 1 la primera voz, 2 la segunda y 3 la tercera mientras que 0 indica si suena o no música. La A valdrá 0 si no se está interpretando música. En caso contrario valdrá –1.

```
10 'BLUES 6/4
20 GOSUB 100
30 CLEAR1000
40 A$="t182v15r2.r4l4o4gb-o5l2d.d.
l4d.l8co4ge-l4dcfl2d.d.r2l4ce-gb-o
5l2d-l4d-o4l2b-.b-o5l4e-d-o4b-g-l2
f.fl4e-gl2cc.r4l8a-fa-fl2a-.r4l8b-
gb-gl2b-.l4o5ce-fg.l8fl4e-c.o4l8b-
l4gfgb-"
50 B$="t182v15o5l2d.d.l4d.l8co4ge-
l4dcfl2d.d.r2l4ce-gb-o5l2d-l4d-o4l
2b-.b-o5l4e-d-o4b-g-l2f.fl4e-gl2cc
.r4l8a-fa-fl2a-.r4l8b-gb-gl2b-.l4o
5ce-fg.l8fl4e-c.o4l8b-l4gfgb-"
60 PLAYA$
70 PLAYB$
80 PLAY"t182o5l2d..
90 GOTO 90
100 OPEN "GRP:"AS#1
110 SCREEN 3
120 PSET (40,70):PRINT#1,"BLUES":P
SET (60,120):PRINT#1,"6/4"
130 RETURN
```

```
10 FOR A= 1 TO 2
20 A$="s3 m7000 c2.o3g8.b16o4c2.o3
g8.b16o4c8c8o3g8b8o4c8c8o3g8b8o4c4
e4g4b4o5c1c+1o4o5d4"
25 B$="s3 m7000 c2.o3g8.b16o4c2.o3
g8.b16o4c8c8o3g8b8o4c8c8o3g8b8o4c4
e4g4b4o5c1c+2.l16agfed2"
30 PLAY A$:PLAY B$
40 NEXT
```

```
10 'motivo sinfonía JUPITER(W.A. M
OZART)
20 A$="s3m7800c4r8o3g18a18b18c4r8o
3g18a18b18c4r2r8o5c8c4.b8d4.c8g2f4
"
30 PLAY A$
```

SOUND Y BEEP

Hemos estado viendo hasta ahora el funcionamiento del MML y de la potentísima instrucción PLAY que poseen todos los MSX. Veremos ahora cómo también es posible acceder directamente a cada uno de los registros del Generador Programable de Sonido (PSG).

Este generador dispone de 14 registros o canales, numerados del 0 al 13 de un byte cada uno y en los que, lógicamente, se pueden almacenar valores comprendidos entre 0 y 255. A la vista de los dicho hasta ahora, es evidente que la sintaxis de SOUND es la siguiente:

Funciones de los regitros del PSG y gama de datos de escritura		
N.º de registro	Función	Gama de datos
0	Frecuencia del canal A	0–255
1		0–15
2	Frecuencia del canal B	0–255
3		0–15
4	Frecuencia del canal C	0–255
5		0–15
6	Frecuencia de ruido	0–31
7	Selecciona un canal para generación de tonos y ruido.	0–63
8	Volumen del canal A	0-15 La variación del volumen ocurrirá cuando se seleccione 16.
9	Volumen del canal B	
10	Volumen del canal C	
11	Frecuencia del patrón de variación de volumen.	0–255
12		0–255
13	Selección del patrón de variación de tonos y ruido.	0–14

SOUND registro, datos

Como acabamos de ver en PLAY, podemos emitir sonidos por tres canales simultáneamente: Llamemos a cada una de las voces A, B y C respectivamente. Los catorce registros del PSG también nos permiten hacer esto. Veamos qué hace cada uno de los registros.

Los registros 0 y 1 corresponden a la voz A. El registro 0 es el byte de orden inferior y el 2 el de orden superior. No se utilizan los 4 bits superiores de estos bytes. Los registros 2 y 3 corresponden a la voz B. Corresponden respectivamente a los bytes de menor y mayor peso y tampoco se utilizan los 4 bits superiores de estos bytes.

Lo dicho hasta ahora resulta igualmente válido para los registros 4 y 5. Es decir, el registro 4 es el byte de menor peso de la voz C y el 5 el de mayor; y tampoco se utilizan los 4 bits superiores de estos bytes. En los registros 1, 3 y 5 se pueden escribir valores oscilantes entre 0 y 15 y en los registros 0, 2 y 4 valores entre 0 y 255.

El registro 6 es el generador de ruido y produce «ruido blanco» con el que se pueden obtener interesantes efectos sonoros. Almacena valores comprendidos entre 0 y 31 que corresponden a las frecuencias de ruido.

El registro 7 es el mezclador. Selecciona la fuente del sonido (uno de los tres canales o el generador de ruido) y los conecta o apaga. Los dos bits de orden superior (los dos últimos) no se usan. Los sexto, quinto y cuarto activan el ruido y los tres últimos los tonos. Puede almacenar datos comprendidos entre 0 y 63. Para tener una visión más clara del resultado que queremos obtener con este registro, podemos escribir en el los datos en binario, lo que nos ayuda a ver cuales de los canales conectamos o apagamos. Por ejemplo:

```
SOUND 7,&B00111000
```

Debemos tener en cuenta que el orden de las voces es CBA, siendo la voz A la menos significativa.

Los registros 8, 9 y 10 controlan el volumen de salida de los sonidos producidos por los tres canales: El registro 8 esta asociado al canal A, el 9 al canal B y el 10 al canal C. Cada uno de

estos registros puede almacenar valores comprendidos entre 0 y 16, correspondiendo el 0 al silencio y 15 al máximo volumen. Cuando el valor sea 16 daremos paso a la variación de volumen para patrones de envolvente.

Los canales 11 y 12 son los que nos permiten regular el periodo del envolvente, que se determina mediante los datos que escribamos en los registros 11 y 12. Podemos obtenerlos de acuerdo con la siguiente fórmula:

$$\frac{1.996.750\ (Hz)}{\text{periodo}\ (Hz)*256} = 256*\ (\text{datos reg. } 12) + (\text{datos reg. } 11)$$

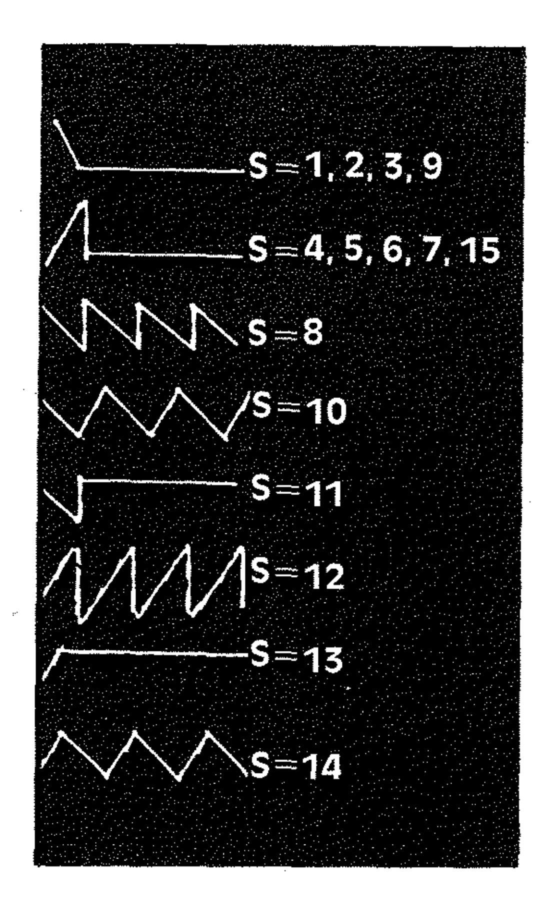
En ella podemos darnos cuenta de que el registro 11 es el byte de orden inferior del periodo de envolvente y el registro 12 el de orden superior.

Si por ejemplo deseamos establecer el periodo a 15 Hz deberemos escribir 8 en el registro 11 y 2 en el 12:

$$1.996.750/(256*15) = 256 * \underline{2} + \underline{8}$$

Tras esto, sólo nos queda establecer 16 como volumen del canal en el que deseemos utilizar el patrón calculado. El registro 13 es el que nos posibilita la selección de una de las ocho formas de envolvente disponibles (sólo utiliza los cuatro bits inferiores). Puede contener valores entre 1 y 14. El gráfico de la página siguiente muestra las 8 formas de envolvente.

```
10 'Despegue de helicoptero
20 SOUND 1,12
30 SOUND 4,222
40 SOUND 8,10
50 SOUND 2,33
60 SOUND 9,110
70 SOUND 7,&B10001111
80 SOUND 3,255
90 SOUND 7,&B10000101
100 FOR Z=0 TO 255:SOUND 13,Z:NEXT
110 FOR S= 1 TO 255:SOUND 11,S:NEXT
120 FOR C= 1 TO 1000:FOR A=0 TO 254:SOUN
D 6,A:NEXT:NEXT
130 SOUND 11,2
140 SOUND 2,33
150 SOUND 9,0
```



Todavía nos queda otra sentencia para producir sonido, aunque es la más limitada de todas. Se trata de BEEP, que emite una señal acústica y reinicializa todas las instrucciones sonoras, es decir, para cualquier sonido que se esté produciendo. Equivale a teclear ?chr$(7) o CTRL G y también es producido por cualquier mensaje de error o cuando encuentra STOP durante la ejecución de un programa.

CAPITULO X

Las interrupciones

A menudo resulta muy interesante poder ordenar al microprocesador que interrumpa el trabajo que esta efectuando, en provecho de una tarea de mayor prioridad.

Hasta ahora las rutinas de interrupción que incorporaban los ordenadores personales, sólo eran accesibles en lenguaje máquina. El sistema MSX, sin embargo, nos posibilita el modo de manejar desde el BASIC estas interrupciones asíncronas, independientes del ciclo del reloj interno, o lo que es lo mismo, que se puedan producir en cualquier momento a voluntad del programador. Las posibilidades de controlar estas interrupciones son cinco.

La primera de ellas ya la hemos visto en el capítulo en que hablamos de los SPRITES. Nos referimos a ON SPRITE GOSUB (número de línea). Como vimos, esta instrucción produce una interrupción del programa cuando se solapan (tienen algún punto de intersección) dos SPRITES, en ese momento la máquina produce un salto a la subrutina que comienza en el número de línea indicado en la instrucción.

Las restantes interrupciones son:

ON KEY GOSUB (n de línea), la interrupción se produce cuando se oprime determinada tecla de función.

ON STOP GOSUB (n de línea), cuando pulsamos la tecla de control y la de STOP a la vez, el programa salta a la subrutina indicada.

ON STRIG GOSUB (n de línea), en este caso la barra espaciadora o el disparador del joystick accionan la interrupción.

ON INTERVAL GOSUB (n de línea), genera una interrupción transcurrido un cierto tiempo desde la ejecución de un RUN, ejecutandose cada fracción de tiempo. Este tiempo es igual a N*1/50 de segundo, o sea, que para N=50 el intervalo es de un segundo.

Para que entren en juego estas cinco posibilidades de interrupción necesitan de una sentencia de salida. Esta instrucción es similar en los cinco casos y toma la forma siguiente: SPRITE ON, KEY (número de tecla de función) ON, STOP ON, STRIG (número de mandos para juegos) ON, INTERVAL (N, intervalo de tiempo) ON, repectivamente.

En el caso de KEY (número de tecla) ON es muy importante que no la confundamos con la instrucción KEY ON sin número de tecla de función, pues esta instrucción no tiene ningún tipo de valor en el capítulo de interrupciones y se limita a visualizar o borrar de la pantalla la cadena asignada a las teclas de función.

Todas estas sentencias de constatación de interrupción se complementan con otras dos instrucciones, cada una de las cuales obtienen al sustituir la palabra ON por las palabras STOP, y OFF, con lo que obtendremos, en el caso de la interrupción producida por las teclas de control +STOP las siguientes instrucciones: STOP ON, STOP OFF, y STOP STOP. En el caso de STRIG (mando de juegos) nos da

STRIG (número de mando) ON
STRIG (número de mando) OFF
STRIG (número de mando) STOP

y lo mismo para las restantes. En el fondo estas instrucciones cumplen la misma función, aplicada a distintos tipos de interrupción.

El funcionamiento de estas instrucciones no es en principio complicado.

La primera sentencia del tipo ON KEY (n) GOSUB (número de línea) nos indica que se puede producir una interrupción (insistamos que puede producir, no que se vaya a producir) y que en caso de que se produzca, esa interrupción envía el flujo del programa a un número de línea determinada.

La siguiente instrucción del tipo KEY (n) ON confirma la

anterior orden y es en ese momento, cuando el ordenador empieza a investigar si se cumple la condición de interrupción (apretar una tecla, la barra de espacios, el disparador de joystick, etc), y cuando esto se cumple se empieza a ejecutar la subrutina cuyo inicio está en la línea definida enla primera instrucción ON KEY GOSUB (n). Por ejemplo, si durante un programa preparamos una interrupción definida mediante ON KEY GOSUB 1000 y la activamos mediante KEY (1) ON, en cuanto pulsemos la tecla de función número 1 el flujo del programa salta a la línea 1000 y ejecutan las instrucciones que encuentre hasta la instrucción RETURN, que devuelve el control a la línea siguiente a la que se encontraba antes de pulsar la tecla de función. Si durante la ejecución de la subutina se vuelve a dar la condición de interrupción, mediante la tecla de función número 1 (o mediante otro dispositivo de interrupción previamente definido) la subrutina sigue ejecutándose y al regresar tras el RETURN al programa principal, antes de continuar vuelve a ejecutar la subrutina de interrupción. Es decir que identifica la nueva condición de interrupción y se reserva su ejecución para cuando haya terminado de ejecutar la primera.

```
10 ON KEY GOSUB 1000
20 KEY (1) ON
30 SCREEN 2
40 LINE (30,90)-(100,250),,B
50 GOTO 40
1000 REM SUBRUTINA DE INTERRUPCION
1100 CLS
1200 FOR I=10 TO 100 STEP 10
1300 CIRCLE (130,98),I,,,1,.71
1400 NEXT I
1500 CLS
1600 RETURN
```

Evidentemente, se pueden forzar interrupciones dentro de una rutina de interrupción (interrupción anidada). Para ello basta con introducir dentro de esta otra instrucción del tipo KEY (n) ON, con lo cual si se produce una nueva condición, la rutina se ejecuta otra vez desde el principio sin esperar al final y luego sigue con la primera interrupción de forma similar a la técnica de los bucles anidados.

```
10 ON KEY GOSUB 60
20 KEY (1) ON
30 SCREEN 2
40 LINE (30,90)-(100,250),,B
50 GOTO 40
60 REM SUBRUTINA DE INTERRUPCION
70 KEY (1) ON
80 BEEP:CLS
90 FOR I = 10 TO 90 STEP 10
100 CIRCLE (130,98),I,,,4,.71
110 NEXT I
120 CLS
130 RETURN 40
```

Si por el contrario queremos, que una vez ejecutada la primera interrupción no sean posibles nuevas interrupciones nos basta con introducir una instrucción del tipo KEY (n) OFF con lo cual el ordenador ignora la condición impuesta. Finalmente, con las instrucciones del tipo KEY (n) STOP se consigue que reconozca la condición impuesta, pero no ejecuta la interrupción hasta que más adelante se le de una orden del tipo KEY (n) ON.

Dado que exiten 10 teclas de función (incluyendo mayúsculas) cinco posibles disparadores de joystick (que van del 0 al 4), si añadimos las posibilidades de INTERVAL SPRITE y STOP nos encontramos con la posibilidad de gestionar 18 subrutinas de interrupción independientes. Si unimos a éstas la posibilidad de producir interrupciones anidadas dentro de otras, por condiciones diferentes, veremos que las posibilidades son casi ilimitadas, aunque exigen una programación minuciosa. Los programas de la página siguiente nos ilustran al respecto.

Recuerde que cuando se acepta una interrupción, ésta se produce al terminar la ejecución de la sentencia BASIC en curso y RETURN devuelve el flujo del programa a la siguiente sentencia. Además, tenga cuidado si utiliza un valor para «INTERVAL ON» muy pequeño, pues es posible que nunca llegue a salir de la rutina de interrupción (su valor máximo, puede ser 65535).

```
10 ' esto es un ejemplo
20 ' de on sprite gosub
30 ' cuando los sprites
40 ' se solapan oiremos
50 ' oiremos un beep.
60 SCREEN 2
70 A$=CHR$(60)+CHR$(62)+CHR$(71)+CHR$(207)
80 B$=CHR$(63)+CHR$(126)+CHR$(124)+CHR$(56)
90 SPRITE$(0)=A$+B$
100 ON SPRITE GOSUB 170
110 SPRITE ON
120 FOR X=0 TO 255
130 PUT SPRITE 0,(X,100),15,0
140 PUT SPRITE 1,(255-X,100),10,0
150 NEXT X
160 END
170 SPRITE OFF
180 BEEP
190 SPRITE ON
200 RETURN
```

```
10 ' ejemplo de on strig gosub
20 ' cuando pulse la barra de
30 'espacios,aparecera hola,
40 'al no haberintroducido
50 ' ninguna instruccion RETURN
60 'no podemos salir de esta
70 'subrrutina de interrupcion.
80 'si pulsamos el disparador
90 'del primer joistick,aparece
100 'hola de nuevo, y si es el
110 ' el disparador del segundo
120 'joistick, entonces aparece
130 'otro mensaje.
140 ON STRIG GOSUB 170,190,210
150 STRIG(0) ON:STRIG(1) ON:STRIG(2) ON
160 GOTO 150
```

```
170 PRINT "Hola"
180 GOTO 170
190 PRINT "Hola de nuevo"
200 GOTO 190
210 PRINT "vaya,que pesado"
220 GOTO 210
```

```
10 ON STOP GOSUB 100
20 STOP ON
30 PRINT "programa principal"
40 GOTO 40
50 PRINT "ctrl+stop ejecutado"
60 RETURN 30
```

```
10 ON INTERVAL=50 GOSUB 100
20 INTERVAL ON
30 SCREEN 2,1
40 A$= CHR$(&H18)+"<"+"~"
50 B$= CHR$(&HFF)+CHR$(&H18)
60 C$= CHR$(&H18)+CHR$(&H18)
70 D$= CHR$(&H18)
80 SPRITE$(1)=A$+B$+C$+D$
90 GOTO 90
100 X=INT(RND(1)*256):Y=INT(RND(1)*192)
110 C=INT(RND(1)*14)+2
120 PUT SPRITE 1,(X,Y),C,1
130 RETURN 90
140 LLIST
```

CAPITULO XI

Introducción al lenguaje máquina

En esencia lo único que puede hacer un ordenador es mover números de un lugar a otro y efectuar las más básicas operaciones matemáticas (suma y resta). Visto asi parece sorprendente que se los describa como aparatos potentes y capaces de realizar un gran número de misiones.

Una de las razones que lo hace posible es que los micros domésticos son suministrados con un sistema que nos permite decirles qué tienen que hacer en un lenguaje que se aproxima mucho más al inglés que al idioma que habla el propio aparato. Programamos los aparatos en BASIC, un lenguaje de ordenador diseñado para hacer la programación general bastante simple.

Efectivamente, el lenguaje BASIC es el medio para llegar a un final y el final es la producción de un código que el ordenador entiende y que le hace reaccionar de la manera que queríamos originalmente. Pero el ordenador no sabe nada de BASIC, nada de variables y muy poco de cualquier cosa que pudiéramos considerar útil. Habla un lenguaje completamente diferente, extremadamente simple, llamado **código de máquina.**
El código de máquina consiste en una corriente de números binarios (0 y 1) que el aparato examina y que después le hacen reaccionar en una manera que depende del diseño de uno de los chips que están en el interior del ordenador.

Este chip es el microprocesador, que en todos los ordena-

dores MSX es el Z80. Este procesador es uno de los chips más versátiles de su género.

Cuando programamos un micro MSX en BASIC sigue necesitando recibir sus instrucciones en su propio código de máquina, que es único para los Z80 e ininteligible para cualquier otro microprocesador.

¿Cómo sabe el ordenador cómo reaccionar a las instrucciones BASIC que le damos? De la misma manera que intentaríamos entender a alguien que nos hablara en arameo a través de un intérprete.

En términos del ordenador, un intérprete es un programa que convierte las instrucciones del lenguaje que usamos al código de máquina que emplea el procesador. Como esta cuestión es bastante complicada y tiene que ser llevada al cabo con un máximo de eficacia tanto en términos de velocidad como en términos del uso de memoria, el intérprete está generalmente escrito en el propio código máquina. Un intérprete BASIC examina cada instrucción en particular que tecleamos y después ejecuta un conjunto predeterminado de instrucciones en código de máquina para llevar a cabo la función deseada.

Un intérprete no es muy inteligente y es de hecho incapaz de recordar la mayoría de las cosas que ha examinado anteriormente, tanto es así que tiene que hacer exactamente la misma cosa una y otra vez.

Esto hace que la interpretación sea muy lenta. Aunque un programa que escribimos parece funcionar a una velocidad tremenda, en términos de lo que es capaz el aparato, va muy lentamente. Así que ¿qué significa esto? Simplemente que si quiere escribir un programa muy rápido, BASIC no le servirá. Empleando el BASIC significa también que muy a menudo muchas de las posibilidades de «bajo nivel» del aparato no están disponibles para nosotros a través del intérprete.

Un ejemplo ideal para ilustrar esto es un programa de juego donde estamos intentado mover un objeto fácil y velozmente a través de cierta área de la pantalla. La velocidad no puede ser realizada en BASIC y no hay instrucciones en BASIC que nos permitan llevar a cabo este tipo de operación. En este tipo de situaciones estamos invariablemente forzados a programar en el mismo código de máquina.

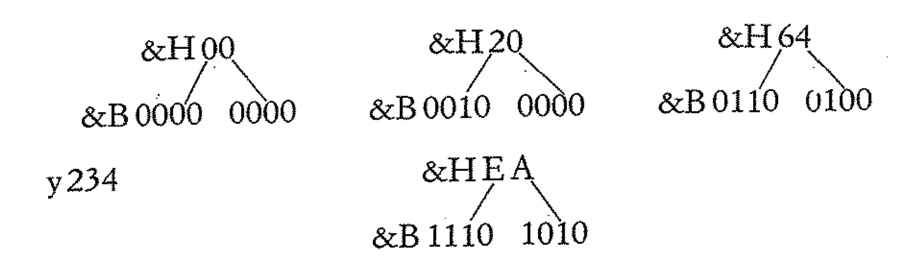
Acabamos de ver que el código de máquina no es otra cosa

que una serie de números, así que a primera vista puede parecer que la programación en código de máquina no va a ser una ocupación muy divertida. Lo cual no deja de ser cierto, pues, una auténtica programación en código de máquina, nos exige conocer a fondo el funcionamiento del microprocesador, que es más de lo que se propone este libro. Sin embargo el BASIC MSX dispone de un gran número de instrucciones tendentes a facilitar esta labor, así como otras, que nos permiten, con los conocimientos necesarios, manipular directamente en las áreas de memoria donde están ubicados, datos e incluso instrucciones del programa. Efectivamente, la memoria de un ordenador se reparte entre dos grandes bloques, la RAM y la ROM. Tanto uno como otro están compuestos por celdillas perfectamente numeradas y ordenadas que el microprocesador controla. Estas celdillas están direccionadas por un número exclusivo para cada una y suele utilizarse notación HEXADECIMAL para numerar estas direcciones.

El principal interés de la notación HEXADECIMAL reside en la facilidad para transformarla en binaria de un vistazo, sin necesidad de cálculos aritméticos, mientras es evidente la dificultad que entraña pasar de decimal a binario. La traducción de HEXA a binario es directa según la siguiente tabla de conversión:

0	00000000	00
1	00000001	01
2	00000010	02
3	00000011	03
4	00000100	04
5	00000101	05
6	00000110	06
7	00000111	07
8	00001000	08
9	00001001	09
10	00001010	0A
11	00001011	0B
12	00001100	0C
13	00001101	0D
14	00001110	0E
15	00001111	0F

De la siguiente manera traducimos Hexadecimal o binaria; los números decimales 0,32 y 100



Con la notación decimal el número 65535 no nos salta a la vista ni se nos presenta como importante dentro de la estructura del microprocesador. Sin embargo, si lo traducimos a Hexadecimal nos encontramos con

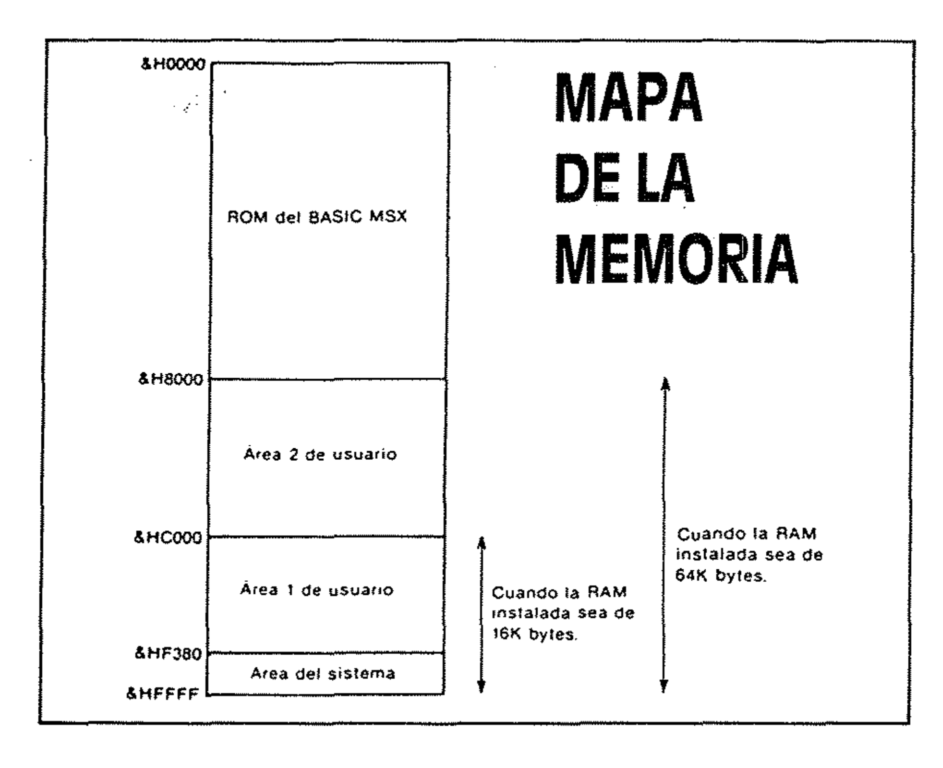
&H FFFF = 65535

que es el tope máximo de memoria que puede direccionar el microprocesador Z80A. El kilo de los informáticos, la celebre K, se nos convierte en &H400 o sea 1024 en decimal (precisamente es 1024 para que sea redondo en HEXA; es muy importante no pensar que el Kbyte es &H1000 lo cual serían 4 K).

Mediante el programa inserto al final de este capítulo podemos observar el contenido de las áreas de memoria que deseemos (introduciendo una dirección de inicio en Hexadecimal).

Así pues, la memoria de un ordenador MSX se numera por áreas desde &H0000 (Hexadecimal 0) hasta &HFFFF, en un aparato de 64Ky se reparte de la siguiente manera (ver gráfico)

MAPA DE MEMORIA	
&HFFFF	AREA DE TRABAJO SISTEMATICO
&HF380	BLOQUE DE CONTROL DE ARCHIVO
	AREA DE CADENAS
	AREA DE APILADO
	AREA LIBRE
	VARIABLE
&H8000 &HC000 para 16K MSX	AREA DE PROGRAMA BASIC
&H0000	ROM DE MSX BASIC



CONFIGURACION DEL AREA DEL USUARIO

Area de programa
Almacena un programa con sus números de línea.

Area de variables
Almacena datos numéricos y los punteros para los datos alfanuméricos.

Area de variables de matriz
Almacena los datos de las variables de matriz. También almacena el puntero para el área de cadenas si los datos son alfanuméricos.

Area libre
No utilizada. El tamaño podrá conocerse utilizando la función FRE.

Area de la pila
Utilizada para guardar direcciones de retorno.

Area de cadenas
Almacena cadenas incluyendo las variables y variables de matriz alfanuméricas.
El tamaño podrá especificarse con una sentencia CLEAR.

Bloque de control de archivos
Utilizado durante la entrada/salida de archivos.

Dirección	Área
&H8000 (&HC000)	Área de programa
	Área de variables
	Área de variables de matriz
	Área libre
	Área de la pila
	Área de cadenas
&HF37F	Bloque de control de archivos

de &H0000 a &80000 se encuentra la ROM o memoria de solo lectura, pues no podemos alterar su contenido (esta memoria no desaparece al desenchufar el ordenador). Es en esta memoria ROM donde se encuentra el traductor de BASIC a lenguaje máquina y las rutinas necesarias para el funcionamiento del sistema, rutinas de I/O, gestión de periféricos, etc. etc. De la &H80000 a la &HF380 (siempre en el caso de un aparato con 64K), disponemos de espacio para introducir programas en

BASIC, y desde allí a la &HFFFF, es la zona del sistema en que se almacenan los valores de las variables, las cadenas y tablas, así como los espacios para dispositivos de archivo que creamos con la instrucción MAXFILES, y es llamada área de trabajo del sistema. Este espacio se puede modificar con la instrucción CLEAR. Estas dos últimas divisiones de la memoria reciben el nombre de RAM (Random Acces Memory) memoria de acceso aleatorio, pues en ellas podemos grabar y leer, aunque por ello también su contenido desaparece al desconectar el aparato (RESET).

Mediante la instrucción CLEAR podemos reservar en la RAM un espacio para rutinas en código máquina.

Ya vimos la instrucción CLEAR en el capítulo sobre series y tablas, aunque aquí nos interese su segundo parámetro, que allí no estudiamos.

Con CLEAR se puede reservar espacio de memoria para variables de cadena y código de máquina en el área de trabajo de sistema.

Se puede reservar espacio de memoria cambiando el área de memoria más alto disponible para el uso de BASIC. El área de memoria más alta por defecto es &HF380 pero puedes bajarla usando CLEAR,(men-alt). El espacio de memoria creado puede ser usado para almacenar programas en código de máquina y datos.

El área de memoria entre &HFFFF y &HF380 es usado por el ROM de BASIC como espacio de trabajo de sistema.

Recuerda que CLEAR (espacio de cadena) decide el tamaño del espacio para cadenas.

SYNTAXIS

CLEAR ⟨espacio de cadena⟩	Decide el tamaño del espacio para las cadenas
CLEAR, ⟨mem-alt⟩	Decide el lugar de memoria más alto disponible para BASIC (la coma , anula el 1.er parámetro)
CLEAR ⟨espacio de cadena⟩ CLEAR, ⟨mem-alt⟩	Decide el tamaño del espacio de cadena como también el lugar más alto de memoria disponible para BASIC

Ejemplos:
CLEAR limpia el área de variables, borrándolo todo.
CLEAR 255 aumenta el área de trabajo de cadena a 255 bytes.
CLEAR, &HF300 crea espacio para el programa en código de máquina del usuario entre &HF300 y &HF380.

Existen en el BASIC MSX una serie de instrucciones que nos facilitan la labor de introducir programas en lenguaje máquina y su posterior ejecución, asimismo, es posible, el introducir rutinas en lenguaje máquina y ser ejecutadas desde un programa en BASIC, sin embargo la programación en código máquina, requiere unos conocimientos de la arquitectura interna del ordenador, y sus registros, así como alguna experiencia en lenguaje ensamblador, por lo que comentaremos sin entrar en pormenores estas sentencias y funciones.

Mediante DEFUSR sintaxis DEFUSR (A) = (dirección memoria)

Podemos introducir en la dirección expresada una serie de las rutinas en lenguaje máquina que irán numeradas de 0 a 9 [el número lo colocaremos en (A)] la dirección es, naturalmente, la dirección de inicio de la subrutina, pues ésta tiene ya el final marcado mediante un código END o de final de programa. Como nada nos prohibe introducir una dirección de memoria de la ROM, con el suficiente conocimiento, del FIRMWARE podemos utilizar las rutinas del sistema operativo, en nuestro provecho, durante la ejecución de nuestros programas en BASIC.

Para llamar a la ejecución a cualquier subrutina definida por DEFUSR, necesitaremos utilizar una función USR sintaxis x = USR (A) (expresión) en la que A es el número que le hemos dado a la subrutina en la instrucción DEFUSR, y la expresión, es la evaluada mediante esta subrutina, es decir, el valor de la expresión, pasa a ser argumento del la subrutina en máquina (dicho de otra manera, el valor de la expresión, sera el manipulado, mediante la subrutina definida).

Ejemplo, si mediante una sentencia DEFUSR preparamos la entrada a la subrutina del sistema operativo que se encuentra alojada en la dirección &H003E y luego la ejecutamos mediante un USR repondremos todos los valores de las teclas de fun-

ción a sus valores por defecto (los de la puesta en marcha). Para comprobarlo, primero modifica los valores de las teclas de función y a continuación ejecuta estas líneas de programa.

```
10 DEFSR3 = &H003E
20 x = USR3(x)
```

En este caso la expresión x sólo tiene una valor sintáctico, pues en la subrutina ejecutada no intervienen calculos ni operaciones de ningún tipo. Sin embargo, obsérvese la forma de utilizar USR, que por otro lado es típica de una función.

Es decir en la línea 20 decimos que X es igual al resultado de modificar el contenido primario mediante la subrutina 3 escrita en código máquina y definida en la línea 10. Para volver al BASIC se utiliza una sentencia RET.

Las dos instrucciones más importantes para trabajar en código máquina son, sin embargo PEEK y POKE, con las cuales podemos escribir y leer directamente, los contenidos de un área de memoria, mediante su dirección y de esta forma escribir los programas que ejectutaremos con USR.

POKE (dirección de memoria) (VALOR)

POKE y PEEK son dos sentencias del BASIC íntimamente relacionadas con el lenguaje máquina.

La sentencia POKE, como hemos visto en su sintaxis está formada por dos valores separados por una coma (,). El primer valor es la dirección de memoria que deseemos modificar y tiene que ser un valor numérico comprendido entre 0 y 65535.

El segundo valor representa el contenido que deseemos introducir en esa dirección. Naturalmente, cada dirección es un único byte que puede tomar 256 (2^8) valores diferentes que constituye la unidad de almacenamiento de datos más pequeña accesible al usuario. ¿De dónde surgen estas 256 posibilidades? Nada más sencillo:

Sabemos que cada byte está compuesto por 8 bits. Imaginemos que estos bits están ordenados de menor a mayor (de 0 a 7) y que cada uno de estos ordinales corresponde a una potencia de 2. Es decir, el primer bit será 20, por lo tanto igual a 1, el segundo 2^1 (2); el tercero 2^2 (4)...

0	1	2	3	4	5	6	7
1	2	4	8	16	32	64	128
2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7

Por lo tanto, si hacemos un poke a una dirección de memoria cualquiera, y ponemos como contenido de esa dirección 1, querrá decir que hemos encendido el primer bit de ese byte, de tal manera; si sabemos qué bits de cada byte queremos encender o apagar), simplemente debemos calcular la potencia a que debe ser elevado 2, y en su caso (si deseamos encender o apagar varios) realizar las sumas pertinentes. Si por error diéramos un valor mayor que 255 obtendríamos «Illegal function call».

La función PEEK nos da el contenido de la dirección de memoria que especifiquemos entre paréntesis. Por ejemplo PEEK(65535). En esencia, es la herramienta que nos permite ver el contenido de cualquier posición de memoria deseada.

¡Cuidado con POKE!. Si toca un área del sistema o se introduce un contenido incorrecto en una dirección puede bloquear el programa, y sólo podrá solucionarlo haciendo RESET.

VPEEK y VPOKE

Corresponden exactamente a PEEK y POKE, pero afectan a la RAM de Vídeo o VRAM que dispone de 16 K. independientes para gestión de pantalla.

Gracias al chip de vídeo TMS-9924A podemos realizar una gran cantidad de operaciones sin ocupar memoria RAM del sistema.

La sintaxis es similar a PEEK o POKE

X = VPEEK (dirección)
VPOKE (dirección), (VALOR)

y los datos máximos que podemos dar a dirección y valor son 16385 y 255 respectivamente, pues sólo disponemos de 16 K.(16383 bits), y de una capacidad de almacenaje de un byte (valor máximo contenido en un byte = 255).

La instrucción BASE y la VDP (que no estudiaremos), completan las utilidades de VPEEK y VPOKE.

Podemos ver sencillos ejemplos de estas instrucciones en las carteleras de presentación de algunos de los programas de música que damos a título de ejemplo.

```
1 'MSX
5 REM si se desea parar la pantalla oprimir stop, son otro stop la ejecucion continuá.
10 SCREENJ:COLOR 1,14:KEY OFF
20 WIDTH 35
30 INPUT "Dirección inicial (en decimal)";D
40 IF D<0 OR D>65535!THEN GOSUB 240
50 D$=STRING$(4-LEN(HEX$(D)),"0")+HEX$(D)
60 PRINT D$;" ";
70 FOR I=0 TO7
80 II=PEEK(D+I):II$=STRING$(2-LEN(HEX$(II)),"0")+HEX$(II)
90 PRINT II$;" ";
100 NEXTI
110 FORI=0 TO 7
120 II=D+I
130 II=PEEK(II)
140 IFII<32 THEN II=46
150 IF II>126 THEN II=46
160 PRINT CHR$(II):
170 NEXT I
180 PRINT
190 D=D+8
200 D$=INKEY$
210 IF D$=" " THEN 30
220 GOTO 50
230 END
240 PRINT:PRINTTAB(8)"*";"Introducción errónea";"*"
250 GOSUB 260:RETURN
260 PRINT:PRINTTAB(10)"Pulsa una tecla"
270 Z$=INKEY$:IF Z$="" THEN 270
280 RETURN
```

PROGRAMAS

ALFABETICO

```
10 SCREEN0:CLS:COLOR 1,14,14
20 LOCATE 5,1:PRINT"Este juego consiste
en"
30 LOCATE 5,2:PRINT"ordenar alfabeticame
nte":
40 LOCATE 5,3:PRINT"la serie de letras q
ue aparece."
50 LOCATE 5,4:PRINT"Para ello te servira
s de"
60 LOCATE 5,5:PRINT"los numeros que hay
debajo"
70 LOCATE 5,6 :PRINT"de cada letra.Debes
 introducir"
80 LOCATE 5,7 :PRINT"un numero de 2 cifr
as:"
90 LOCATE 5,8 :PRINT"La primera debe ser
 la de debajo"
100 LOCATE 5,9 :PRINT"de la letra que va
s a sustituir"
110 LOCATE 5,10:PRINT"la segunda, la de
la letra que"
115 LOCATE 5,11:PRINT"sustituyes."
120 LOCATE 5,12:PRINT"La partida finaliz
a "
130 LOCATE 5,13:PRINT"cuando las has ord
enado"
140 LOCATE 5,14:PRINT"correctamente toda
s."
```

```
150 LOCATE 7,16:PRINT"PULSA UNA LETRA"
160 Z$=INPUT$(1):CLS
180 CLS:INPUT"entra un numero:":W
190 FORI=1TOW:Y=RND(1):NEXT
200 SCREEN1:COLOR15,13,11:KEYOFF:CLS
210 A$=""
220 FORI=1TO10
230 U$=CHR$(INT(RND(1)*10+65))
240 FORJ=1TOI-1
250 IFU$=MID$(A$,J,1)THEN230
260 NEXTJ
270 A$=A$+U$
280 NEXTI
290 S=1
300 LOCATE0,4:PRINT"resultado:";A$
310 PRINTTAB(10);"0123456789"
320 LOCATE0,9:PRINT"intento numero";S
330 LOCATE33,12:PRINT"          "
340 IFA$="abcdefghij"THENPRINT"encontrad
o en";S;"intentos":PLAY"abc":END
350 LOCATE0,12:INPUT"inversion (forma a,
b):";R$
360 I=ASC(LEFT$(U$,1))-47
370 J=ASC(RIGHT$(U$,1))-47
380 GOSUB440
390 IFLEFT$(U$,1)="0"ORLEFT$(U$,1)="9"TH
EN410
400 I=I-1:J=I+2
410 GOSUB440
420 S=S+1
430 GOTO300
440 B$=""
450 FORT=1TOLEN(A$)
460 IFT=ITHENN$=MID$(A$,T,1)
470 IFT=JTHENM$=MID$(A$,T,1)
480 NEXTT
490 FORT=1TOLEN(A$)
500 IFT=ITHENB$=B$+M$:GOTO530
510 IFT=JTHENB$=B$+N$:GOTO530
520 B$=B$+MID$(A$,T,1)
530 NEXTT
540 A$=B$
550 RETURN
```

CANON A TRES VOCES

```
10 '♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫
20 '♫                                 ♫
30 '♫        CANON A TRES VOCES       ♫
40 '♫                                 ♫
50 '♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫
60 CLS
100 A$="l2o4al4o5co4ag#bo5el8edl2c.l8cdl
4eo4bg#el2f.l4feg#bo5el2d#l4d#d#l1o5el2o
4al4o5co4ag#bo5el8edl2c.l8cdl4eo4bg#el2f
.l4feg#bo5el2d#l4d#d#l1o5e"
110 B$="r1r1l2o4al4o5co4ag#bo5el8edl2c.l
8cdl4eo4bg#el2f.l4feg#bo5el2d#l4d#d#l1o5
el2o4al4o5co4ag#bo5el8edl2c.l8cdl4eo4bg#
el2f.l4feg#bo5el2d#l4d#d#l1o5e"
120 C$="r1r1r1r1l2o4al4o5co4ag#bo5el8edl
2c.l8cdl4eo4bg#el2f.l4feg#bo5el2d#l4d#d#
l1o5el2o4al4o5co4ag#bo5el8edl2c.l8cdl4eo
4bg#el2f.l4feg#bo5el2d#l4d#d#l1o5e"
130 PLAY A$,B$,C$
140 PLAY B$,C$,A$
```

MOON GERMS

```
10 '********************************
20 '*                              *
30 '*         MOON GERMS           *
40 '*                              *
50 '*         JOE FARRELL          *
60 '*                              *
70 '********************************
80 GOSUB 180
90 PLAY "O4"
100 A$="T200R2R8L8C#F#A#AO5C#L4C#L8CO5EL
4ER4C#O4A#.L8G#L1G#R2R8O5L8F#D#C#C#O4A#O
5L2C#.L4C#.O4L8A#L2A#L4O5C#L2E.L4A.L8F#L
4F#F.D.C#L8O4G#A#L2A#L4O5C#L4O4A#L1A#"
110 B$="T200R2R8L8C#F#A#AG#L4G#O5CCR4O4G
#F#.L8D#L1D# R2R8O5L8F#D#O4G#L4G#L2G#.L4
G#.L8F#L2F#L4G#O5L2C.L4E.C#.C.O4A#.G#L8D
#F#L2F#L4G#F#L1F#"
120 C$="T200R2R8L8C#F#A#AD#L4D#L8O5CO4GL
4GR4D#C#.O3L8A#L1A#R2R8O5L8F#D#O4D#L4D#L
2D#.L4D#.L8C#L2C#L4D#L2G.L4B.L8G#L4G#GL8
GL4F.DO3L8A#O4C#L2C#L4D#C#L1C#"
130 PLAY"S3M8000","S1M20000","V12"
140 FOR A= 1 TO 2
150 PLAY A$,B$,C$
160 NEXT
170 GOTO 170
180 SCREEN 3:FOR A=509 TO 1022
190 VPOKE A,[RND(1)*111+1]
200  NEXT
210 RETURN
```

BOSSA NOVA

```
10 GOSUB170
20 B$="s1m7000t160r2.ao5ed8e2..r4d
eco4b-8o5c2..r4o4b-o5co4ag8a2..r4g
a"
30 C$="s1m7000t160f+2.r8a2.."
40 C1$="s1m7000t160f+2.."
50 D$="s1m7000t160o5ed8e2..r4deco4
b-8o5c2..r4o4b-o5co4ag8a2..r4ga"
60 D1$="s1m7000t160o5ed8e2..r4deco
4b-8o5c2..r4o4b-o5co4ag8a2..r4ga"
70 E$="s1m6000t160f+1"
80 F$="s1m5500t200r112agf+fo5co4ag
+f+agef+d1r2.14a."
90 PLAY B$
100 PLAY C$
110 PLAY D$
120 PLAY E$
130 PLAY F$
140 PLAY D1$
150 PLAY C1$
160 GOTO 160
170 OPEN "grp:"AS#1
180 SCREEN 3:COLOR 12,7,2
190 LINE (0,150)-(255,150)
200 PAINT(0,191)
210 PSET(45,30):PRINT#1,"BOSSA"
220 PSET (60,70):PRINT#1,"NOVA"
230 RETURN
```

BLUE BOSSA

```
10 '%%%%%%%%BLUE BOSSA%%%%%%%%%%
20 '%                          %
30 '%      Kenny Durham        %
40 '%                          %
50 '%%%%%%%%%%%%%%%%%%%%%%%%%%%%
60 GOSUB 130
70 S$="t255r2."
80 A$="t255s3m35000o4go5g.f8e-8d4c2..o4b
-4a-2o5g4.f8f1f4.e-8d8c4o4b-2..a-4g2o5f4
.e-1e-8e-4.d-8c8o4b-8a-2..g-4g-4f8b-4f8a
-4a-1a-4g8b-4.a-4g8b-2a-8g1.."
90 PLAY S$
100 FOR Q= 1 TO 2: PLAY A$:NEXT
110 GOTO 110
120 DEFINT A-Z
130 Z=1535
140 SCREEN 3
150 FOR A=1 TO Z
160 C=C+77:IF C>255 THEN C=0
170 VPOKE A,C:NEXT
180 RETURN
```

TEMA DE LA SEPTIMA SINFONIA, DE BEETHOVEN

```
10 '♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
20 '♪                            ♪
30 '♪   Tema de la 7a Sinfonía   ♪
40 '♪                            ♪
50 '♪    Ludwig Van Beethoven    ♪
60 '♪                            ♪
70 '♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪♪
80 DEFINT A-Z
90 Z=1535:IF W=2 THEN Z= Z*4
100 R=INT(W*(RND(1)*7)+1)
110 IF W= 2 THEN COLOR R,R+1:IF R+1>15 T
HEN R+1=14
120 SCREEN 3
130 FOR A=1 TO Z
140 C=C+1:IF C>255 THEN C=0
150 VPOKE A,C:NEXT
160 A$="t80s3m27800o4a4a8a8a4a4a4a8a8a4a
4a4a8b8o5c4c4c4c8c8c4r4"
170 B$="t80s3m27800o4f4f8f8e4e4e4e8e8f4f
4f4f8g8a4a4e4f8g8a4r4"
180 C$="t80s3m27800o4d4d8d8c+4c+4c+4c+8c
+8d4d4d4d8d8c4c4c4c8c8f4r4"
190 A1$="t80s3m27800o5c4c8d8e4e4o4b4b8o5
c+8d4d4o4a4a8a8a4a4a4b8o5c+8d4r4"
200 B1$="t80s3m27800o4a4a8a8g+4g+4g4g8g8
f+4f4f4f8f8e4f4e4e8e8d4r4"
210 C1$="t80s3m27800o4f4f8f8e4e4e4e8e8d4
```

```
d4d4d8d8c+4d4o3a4a8a8o4d4r4"
220 A2$="t80s3m27800o4a4a8a8a4a4a4a8a8a4
a4a4a8b8o5c4c4c4c8c8c2"
230 B2$="t80s3m37800o4f4f8f8e4e4e16f16g4
f16e16e8f8f4f4.f16g16g+8a8a4g16a16b-4a16
g16g8a8a4"
240 C2$="t80s3m27800o4d4d8d8c+4c+4o3a4a8
a8o4d4d4d4d8d8c4c4c4c8c8f2"
250 A3$="t80s3m27800o5c4c8d8e4e4o4b4b8o5
c+8d4d4o4a4a8a8a4a4a4b8o5c+8d4r4"
260 B3$="t80s3m37800o4a4a8a8g+4g4g8e16f+
16g8a16g16g8f+8f4f4f8f8e4f4e16f16g4f16e1
6d4r4"
270 C3$="t80s3m27800o4f4f8f8e4e4e4o3a8a8
o4d4d4d4d8d8c+4d4o3a4a8a8o4d4r4"
280 PLAY A$,B$,C$
290 FOR A=1 TO 2:PLAY A1$,B1$,C1$:NEXT A

300 PLAY A2$,B2$,C2$
310 FOR B=1 TO 2:PLAY A3$,B3$,C3$:NEXT B

320 Z$=INPUT$(1)
```

TERCETO DE LA FLAUTA MAGICA, DE W. A. MOZART

```
10 '♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫
20 '♫                                ♫
30 '♫ Terceto de "La flauta mágica"♫
40 '♫                                ♫
50 '♫        W. A. Mozart             ♫
60 '♫                                ♫
70 '♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫
80 GOSUB 230
90 A$=" v15    o4r2.g4g4.b8o5c8.o4b16a8.
o5d16c4o4b4r4o5c4c4o4a8o5d8d8c8c8o4b8b4o
5c4r4c4c4.o4a8b8.a16b8.o5c16o4b4a4r4"
100 B$="s1m28000o4r2.e4e4.f8g8.g16f8.f16
e4d4r4g4a4.a8g4f4f4e4r4e4f+4.f+8g8.f+16g
8.a16g4f+4r4"
110 C$="s1m28000o4r2.c4c4.d8e8.c16f8.d16
g4o3g4r4o4e4f4.f8e4d4d4c4r4o3a4o4d4.d8d8
.d16d8.d16d4d4r4"
120 A1$="s1m30000o5d4d8c8e8c8c8o4b8o5c8o
4a8a4g4r4g4g4.a8b8o5c8d8c8c8o4b8b4.o5c16
o4b16a8g8g4o5c8o4g8g8f8f8e8g8.e16d4"
130 B1$="s1m28000o4b4b8a8o5c8o4a8a8g8a8f
+8f+4g4r4g4g4.l8ggggggfl4f.l8fffl4e.l8ee
ddce.c16o3b4"
140 C1$="s1m28000o4d4e4e4d4.c8c4o3b4r4g4.
g4.f+8f8e8f8e8e8d8d4.d8c8o3b8c4c8o3b4o4
c4o3g4g4r4"
150 A2$="s1m30000o4g4g4.b8o5c8.o4b16a8.o
```

```
5d16c4o4b4r4o5c4o4a8.b16o5c8.d16c4o4b4o5
f2.d4e8.o4b16o5c8.d16c4o4b4o5c2r4"
160 B2$="s1m28000o4d4e4.f8g8.e16f8.f16e4
d4r4g4f8.f16g8.a16g4g4g2.g4g8.f16e8.f16e
4d4e2r4"
170 C2$="s3m28000o3b4o4c4.d8e8.c16f8.d16
g2r4e4f8.d16e8.f16e4d4o3b2.o4f4e8.d16c8.
a16g4o3g4o3c2r4"
180 PLAY"t150","t150","t150"
190 PLAY A$,B$,C$
200 PLAY A1$,B1$,C1$
210 PLAY A2$,B2$,C2$
220 GOTO 220
230 DEFINT A-Z
240 Z=1535
250 SCREEN 3
260 FOR A=1 TO Z
270 C=C+34:IF C>255 THEN C=0
280 VPOKE A,C:NEXT
290 RETURN
```

SCRAPPLE FROM THE APPLE & DONNA LEE

```
10 '$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
20 '$                                $
30 '$   Scrapple from the apple      $
40 '$                                $
50 '$         &Donna Lee             $
60 '$                                $
70 '$        Charlie Parker          $
80 '$                                $
90 '$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
1000 DEFINT A-Z
1010 Z=1535
1020 SCREEN 3
1030 FOR A=1 TO Z
1040 C=C+37:IF C>255 THEN C=0
1050 VPOKE A,C:NEXT
1060 PLAY "v15"
1070 A$="t200o5l8r8d+egf+edo4bo5ear4r8l4
gl8f+g.o6l16do5l9bgel8f+o4bo5eo4b-r4.l4o
5f+gl8f+a.l16f+l9edel4er8o6l8do5l4br8o6c
+o5l8bl9f+gg+"
1080 B$="t200l4ar8l8f+gf+ff+af+gar2"
1090 C$="t200l4ar8l8f+gf+ff+do4ao5c+dr2"

1100 PLAY A$
1110 PLAY B$
1120 PLAY A$
1130 PLAY C$
```

```
1140 W$="t230o6r4.l4el16del8dd-co5b-agf+
o4ao5cdl9e-fe-l8dco4bo5cfaedr4.l4d-o4l8b
o5co4f+gb-o5dfao6co5b-dfaa-d-co4b-ao5ceg
l4fl8cde-gb-o6dd-o5ar4l2o6cl8co5b-ago6co
5b-r4o6e-d-co5b-aab-bo6co5b-agf+ao6cdl9e
-fe-l8dco5baa-f+gfo4ao5cl9ee-dc+"
1150 Z$="r2l8o6co5b-dfagdfegb-gl9o6c+d+c
+l8co5bl4ar4o6l9efel8dd-co5b-agf+o4ao5cd
l9e-fe-l8dco4bo5dfaedr2edc+def+gf+edl9b-
o6co5b-l8agl9fgfec+l4dr4r2r8"
1160 Y$="l8o5eag+aa+ba+bo6cd-cd-co5bb-ag
fl16gfl8efgfedo4g+bo5dfg+o6edd-co5b-agf+
e-dco4b-o5dfagfedl4fr4."
1170 PLAYW$:PLAYZ$:PLAY Y$
1180 Z$=INPUT$(1)
```

THE ENTERTAINER

```
10 '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
20 '@                            @
30 '@       SCOTT JOPLIN         @
40 '@                            @
50 '@      THE ENTERTAINER       @
60 '@                            @
70 '@ Adaptación MSX F.J.Guerrero @
80 '@                            @
90 '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
100  GOSUB 550
110  CLEAR 5000
120 Z$="t130s3m10000l8o4a+a"
130 A$="                  o4b8o5g4o4b8o5g4o4b8
o5g2.r8g8a+8a+8b8g8a8b4f+8a4g2g8r8o4a8a+
8
140 B$="                  o3r4d4r4f4r4e4r4d4r4
d4r4f+r4f4f8r4."
150 C$="          o2g4r4o3d4r4c4r4o2b4r4o3dr
4dr4o2gr2o3d"
160 A1$="o4b8o5g4o4b8o5g4o4b8o5c2.r8e8d8
c+8e8g8b4a8g8e8a2a8r8o4a8a+8"
170 B1$="o3r4b4r4d4r4cr4d4r4a4r4c+r4c4c4
r4"
180 C1$="o2g4r4o3d4r4c4r4o2b4b-4a4r4o3a4
r4o4d4o3d4e4f+4"
190 A2$="o4b8o5g4o4b8o5g4o4b8o5g8g2r8g8a
8a+8b8g8a8b4f+8a4g2g8r8g8a8"
```

```
200 B2$="o3r4b4r4d4r4c4r4d4r4b4r4c4r4b4b
8r4."
210 C2$="o3g4r4d4r4c4r4o2b4r4o3d4r4d4r4o
2g4r2."
220 A3$="o5l8bgal4bl8gagbgal4bl8gagbgal4
bf+8a4g2g8r4."
230 B3$="o3r4d4r4b4r4g4r4c4r4b4r4c4r4d4b
8r4."
240 C3$="o2g4r4f4r4e4r4e-4r4d4r4d4r4g4d4
o1g8r4."
250 A4$="o5d4e8d4o4b8o5c8c+8d4e8d4.g4e8f
+8g8a8b8a8g8a8g2g8r8o4b8o5c8"
260 B4$="o3r4g4r8g8a8a+8b4g4r4dr4e4r4g-4
r8l8bcdedba"
270 C4$="o2g4r4o3d4r4g4r4d4r4c4r4c4r4o2b
4r4o3d4r4"
280 A5$="o5d4e8d4o4b8o5c8c+8d4e8d4.e4f+8
f+4f+4e8c+8o4a8o5d2d8r8o4b8o5c8"
290 B5$="o3g4r8g8a8a+8b4d4r4g4r4f+4r4a4f
+4r2."
300 C5$="o2gr4o3dr4g4r4o2b4b-4a4r4a4r4o3
d4c-4o2b4a4"
310 A6$="o5d4e8d4o4b8o5c8c+8d4e8d4.g4e8f
+8g8a8b8a8g8a8g2g8r8d4"
320 B6$="o3r4d4r8g8a8a+8b4g4r4d4r4c4r4g-
4r4d4r4g-4"
330 C6$="o2g4r4o3d4r4g4r4d4r4c4r4c4r4o2b
4r4g4r4"
340 A7$="o5d4e8g4e8g8e8d8g8b8o6d4o5b8g8d
8e4g4b8a4g2.r8o4a8a+8"
350 B7$="o3e4c4b-4b-4g4b4d4b4c+4e4f+4d4b
4d4d8r4."
360 C7$="o2c4o3c4c+4c+4d4d4o2d4d4o3a4a4o
2d4d4o3g4d4o2g8r4."
370 PLAY"s3m10000","s3m30000","v13"
380 PLAY"t130","t130","t130"
390 PLAY Z$
400 PLAY A$,B$,C$
410 PLAY A1$,B1$,C1$
420 PLAY A2$,B2$,C2$
430 PLAY A3$,B3$,C3$
440 '
450 PLAY A4$,B4$,C4$
```

```
460 PLAY A5$,B5$,C5$
470 PLAY A6$,B6$,C6$
480 PLAY A7$,B7$,C7$
490 '
500 PLAY A$,B$,C$
510 PLAY A1$,B1$,C1$
520 PLAY A2$,B2$,C2$
530 PLAY A3$,B3$,C3$
540 GOTO 540
550 DEFINT A-Z
560 Z=1535
570 SCREEN 3
580 FOR A=1 TO Z
590 C=C+57:IF C>255 THEN C=0
600 VPOKE A,C:NEXT
610 RETURN
```

TECLEE UN NUMERO

```
20 CLS:INPUT"TECLEE UN NUMERO :";F
30 FOR I=1 TO F:Y=RND(1):NEXT
40 SCREEN 1:COLOR 10,1,1:KEY OFF:CLS
50 X=INT(RND(1)*16+5):PLAY"L32N70"
60 PRINT"NUMERO INICIAL:";X
70 PRINT:PRINT"TECLEE 5 OPERACIONES"
80 PRINT:PRINT"DE LA FORMA +5,/2,*4..."
90 F=X
100 DIM R$(5)
110 FOR I=1 TO 5
120 LOCATE0,9:INPUT"TU OPERACION:";R$(I)

130 LOCATE16,9:PRINT"                "
140 A=ASC(LEFT$(R$(I),1))-41
150 IF A<=0 OR A=3  OR A=5 OR A>6THEN 12
0
160 IF A>3  THEN A=(A/2)+1
170 ON A GOSUB 1000,2000,3000,4000
180 LOCATE14,11:PRINT"              "
190 LOCATE0,11:PRINT"NUMERO OBTENIDO:";F

200 NEXT I
210 PRINT:PRINT"OPERACIONES:";R$(4);" ";
R$(1);" ";R$(5);" ";R$(3);" ";R$(2)
```

```
220 PRINT:PRINT"ANIMO!!!..."
230 PLAY"CCCDEDCEDDC":END
1000 F=F*VAL(RIGHT$(R$(I),LEN(R$(I))-1))
1010 RETURN
2000 F=F+VAL(RIGHT$(R$(I),LEN(R$(I))-1))
2010 RETURN
3000 F=F-VAL(RIGHT$(R$(I),LEN(R$(I))-1))
3010 RETURN
4000 F=F/VAL(RIGHT$(R$(I),LEN(R$(I))-1))
4010 RETURN
```

CALENDARIO PERPETUO

```
1 'Calendario Perpétuo
10 SCREEN0:COLOR 15,4:KEY OFF:CLS
20 INPUT"DIA............:";J
30 IF J=0 THEN END
40 PRINT:INPUT"MES............:";M
50 PRINT:INPUT"AÑO............:";A
60 B=A
70 S=INT(A/100)-INT(A/400)
80 IF A<=1582 THEN S=2
90 M$="ENERFEBRMARZABRIMAY JUNIJULIAGOSS
EPTOCTUNOVIDICI"
100 M$=MID$(M$,4*M-3,4)
110 IF M<3 THEN M=M+12:A=A-1
120 S=J-1+INT(13*(M+1)/5)+INT(5*A/4)-S
130 S=S-7*INT(S/7)
140 J$="DOMINGO  LUNES    MARTES   MIERC
OLESJUEVES  VIERNES  SABADO    "
150 J$=MID$(J$,9*S+1,9)
160 LOCATE0,15:PRINT"EL";J;" ";M$;B;"FUE
 UN ";J$:PLAY"CGE"
170 PRINT:PRINT:PRINT"PULSE UNA TECLA":R
$=INPUT$(1)
180 RUN
```

MODIFICACION TABLA DE COLORES SCREEN 1

```
10 PRINT:PRINT:PRINT"Puedes convertir el
 modo de texto 1"
20   PRINT:PRINT"en multicolor utlilizan
do este programa"
30   PRINT:PRINT"Ejecutalo, cuando encue
ntres la combinacion"
40   PRINT:PRINT"de tu gusto, teclea scr
een 0 y despues ? a"
50 PRINT:PRINT"El valor de a, sera el co
ntenido "
60 PRINT:PRINT"de la direccion &HF3BF, q
ue es la que"
70 PRINT:PRINT"has modificado hasta obte
ner el resultado deseado"
80 PRINT:PRINT:PRINT"        PULSA UNA TEC
LA":Z$=INPUT$(1)
100 'screen 1 multicolor
110 SCREEN 1:C=C+1:IF C>15 THEN C=0
120 A= A+1:IF A>255 THEN A=0
130 POKE&HF3BF,A:FOR D= 1 TO 300:COLOR,C
:NEXT:PRINTA:PRINT"ABCDEFGHIJKLMNOPQRSTU
VWXYZ":PRINT"abcdefghijklmnopqrstuvwxyz"
:FOR V= 1 TO 500:NEXT
140 PRINT"1234567890¼½¾∩‰∫√α•□─±[illegible]▼┌":PR
INT"[illegible]▼┌┬┐[illegible]☺♪[illegible]┼┼[illegible]♠♣~☼╳◊└┴┘♂≤≥[illegible]
┌┐[illegible]":FOR V= 1 TO 500:NEXT
150 GOTO 110
```

RECTANGULOS EN 3-D

```
10 '^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
20 '^^^^^^^RECTANGULOS EN 3-D^^^^^^^^^
40 '^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
100 SCREEN 2:COLOR ,1,8
110 FX=RND(1)*255:FY=RND(1)*191
120 DIM PX(3),PY(3),QX(3),QY(3)
130 PX(0)=RND(1)*(.9*255):PY(0)=RND(1)*(
.9*255)
140 B=RND((1)*255-PX(0))/2+50
150 H=RND((1)*191-PY(0))/2+38
160 Q!=RND((1)*.2)+.2
170 PX(1)=PX(0)+B:PY(1)=PY(0):PX(2)=PX(1
):PY(2)=PY(0)+H:PX(3)= PX(0):PY(3)=PY(2)

180 FOR I= O TO 3:QX(I)=PX(I)*(1-Q!)+FX*
Q
190 QY(I)=PY(I)*(1-Q!)+FY*Q!
200 NEXT
210 FOR I=0 TO 3:LINE (QX(I),QY(I))-(QX(
(I+1)MOD4),QY((I+1)MOD4)),15:NEXT
220 FOR I=0 TO 3:LINE(PX(I),PY(I))-(QX(I
),QY(I)),11:NEXT
230 FOR I=0 TO 3:LINE (PX(I),PY(I))-(PX(
(I+1)MOD4),PY((I+1)MOD4)),12:NEXT
240 GOTO130
```

```
10 '^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
20 '^^^^^^^RECTANGULOS EN 3-D^^^^^^^^^
30 '^^^^^CON EFECTO DE DESTELLO^^^^^^
40 '^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
100 SCREEN 2:COLOR ,1,8
110 FX=RND(1)*255:FY=RND(1)*191
120 DIM PX(3),PY(3),QX(3),QY(3)
130 PX(0)=RND(1)*(.9*255):PY(0)=RND(1)*(
.9*255)
140 B=RND((1)*255-PX(0))/2+50
150 H=RND((1)*191-PY(0))/2+38
160 Q!=RND((1)*.2)+.2
170 PX(1)=PX(0)+B:PY(1)=PY(0):PX(2)=PX(1
):PY(2)=PY(0)+H:PX(3)= PX(0):PY(3)=PY(2)

180 FOR I= O TO 3:QX(I)=PX(I)*(1-Q!)+FX*
Q
190 QY(I)=PY(I)*(1-Q!)+FY*Q!+Q!/10
200 NEXT
210 FOR I=0 TO 3:LINE (QX(I),QY(I))-(QX(
(I+1)MOD4),QY((I+1)MOD4)),15:NEXT
220 FOR I=0 TO 3:LINE(PX(I),PY(I))-(QX(I
),QY(I)),11:NEXT
230 FOR I=0 TO 3:LINE (PX(I),PY(I))-(PX(
(I+1)MOD4),PY((I+1)MOD4)),12:NEXT
240 GOTO170
```

JUEGO DE CARACTERES ALFABETICOS EN TODOS LOS MODOS

```
200 COLOR 1,14,10
210 S=0:SCREEN 0
220 GOSUB 330
230 S=1:SCREEN 1
240 GOSUB 330
250 OPEN "GRP:"AS#1
260 COLOR 1,14,7
270 S=2:SCREEN 2
280 GOSUB 340
290 COLOR 1,14,9
300 S=3:SCREEN 3
310 GOSUB 340
320  END
330 FOR I=65 TO 90:LOCATE 5,5:PRINT"SCN"
S;CHR$(1);CHR$(I+32):GOSUB 350:CLS:NEXT:
RETURN
340 FOR I=65 TO 90:PRESET(10,50):PRINT#1
,"SCN"S;CHR$(1);CHR$(I+32):GOSUB 350:CLS
:NEXT:RETURN
350 FOR D= 1 TO 500:NEXT:RETURN
```

JUEGO MATEMATICO (SUMA DE LOS ELEMENTOS DE LAS DIAGONALES)

```
10 DIMA(5,5)
20 INPUTM
30 IFM=0THEN250
40 LETD=1
50 FORI=1TOM
60 FORJ=1TOM
70 READA(I,J)
80 PRINT   A(I,J);
90 NEXTJ
100 PRINT
110 LETD=D+A(I,I)+A(I,M+1-I)
120 NEXTI
130 PRINT
140 IFM/2=INT(M/2)THEN170
150 LETN=INT(M/2)+1
160 LETD=D-A(N,N)
170 PRINT"suma de elementos de las
    diagonales=";D
180 PRINT
190 PRINT
200 RESTORE
210 GOTO20
220 DATA10,11,12,13,14,15,16,17,18,19
230 DATA20,21,22,23,24,25,26,27,28,29
240 DATA30,31,32,33,34
250 END
```

MAS GRANDE MAS PEQUEÑO

```
10 REM***MAS GRANDE / MAS PEQUEÑO***
15 CLS
16 COLOR 1,12
20 PRINT " Este sencillo juego, se inspi
ra en el celebre MASTER-MIND solo que co
plicado por la escasez de pistas,el jueg
o nos pide un numero de cifras de 1 a 7
que compondran el numero a adivinar"
30 PRINT "Para que acertemos el numero e
l ordenador nos da dos pistas, La primer
a es el numero de cifras mayores que en
el numero a adivinar, y la segunda el nu
mero de cifras menores."
40 PRINT " ejemplo : si el numero a adiv
inar es el 2031,y nosotros escribimos 31
40,la maquina nos indica 3 mas grandes e
l 3 el 1 y el 4, y uno mas pequeno,el 0.
 el juego puede ser muy dificil"
50 PRINT:PRINT"      >>>>>PULSA UNA TECLA
<<<<<"
60 Z$=INPUT$(1)
70 CLS:INPUT"TECLEE UN NUMERO :";F
80 FOR I=1 TO F:Y=RND(1):NEXT
90 CLS:INPUT"NUMERO DE CIFRAS:";C
100 IF C>8 THEN 90
110 SCREEN0:COLOR 15,12:KEY OFF:CLS
120 E=1:F=1
```

```
130 X$=STR$(INT(RND(1)*9^C+10^(C-E)))
140 X$=RIGHT$(X$,LEN(X$)-1)
150 LOCATE0,3:PRINT"PRUEBA NO. NRO PRUEB
A       >     <     "
160 LOCATE0,4:FOR J=1 TO 38:PRINT CHR$(2
23);:NEXT
170 LOCATE0,F+4:PRINTTAB(4);E;TAB(10):IN
PUT P$
180 IF P$=X$ THEN 270
190 A=0:B=0
200 FOR I=1 TO C
210 IF MID$(P$,I,1)>MID$(X$,I,1) THEN A=
A+1
220 IF MID$(P$,I,1)<MID$(X$,I,1) THEN B=
B+1
230 NEXT I
240 LOCATE26,F+4:PRINT A;TAB(31);B
250 E=E+1:F=F+1:IF F=18 THEN F=1
260 GOTO 170
270 PRINT:PRINT" HAS ENCONTRADO  ";X$;"E
N ";E;" PRUEBAS":PLAY"Ceg"
280 PRINT:PRINT"OTRA PARTIDA (S/N)":R$=I
NPUT$(1)
290 IF R$="S" OR R$="s" THEN 90
```

POKER

```
10 CLS:PRINT:PRINT:PRINT"En este póker j
uegas contra tu MSX"
20 PRINT:PRINT"Tras el reparto de cartas
 puedes des-":PRINT"cartarte una vez."
30 PRINT:PRINT" Después la máquina te mo
strará su"
40 PRINT:PRINT"jugada y tu mismo verás e
l resultado.":PRINTTAB 10"Suerte"
50 PRINT:PRINT:PRINT:PRINT"PULSA UNA TEC
LA":Z$=INPUT$(1)
60 CLS:INPUT"introduce un numero :":F
70 FORI=1TOF:Y=RND(1):NEXT
80 DIML(16)
90 V$="7 8 9 10J Q K A "
100 C$(0)="♣"
110 C$(1)="♦"
120 C$(2)="♥"
130 C$(3)="♠"
140 SCREEN0:COLOR9,15:KEYOFF:CLS
150 FORI=1TO16
160 L(I)=INT(RND(1)*32)
170 IFI=1THEN210
180 FORJ=1TOI-1
190 IFL(I)=L(J)THEN160
200 NEXTJ
210 NEXTI
```

```
220 K=0:P=0
230 LOCATE0,12:PRINT"cartas rechazadas"
240 GOSUB330
250 IFR$="0"THENPLAY"cde":END
260 FORI=1TOLEN(R$)
270 L(VAL(MID$(R$,I,1)))=L(I+5)
280 NEXTI
290 CLS
300 P=1
310 GOSUB330
320 K=K+10
330 FORI=1TO5
340 J=INT(L(I+K)/8)
350 F=2*(L(I+K)-8*INT(L(I+K)/8))+1
360 R$=MID$(V$,F,2)
370 LOCATE7*I-4,K:PRINTI:PLAY"l64n70"
380 LOCATE7*I-5,K+2:PRINTR$;": ";C$(J)
390 LOCATE7*I-5,K+3:PRINT"-----"
400 NEXTI
410 IFP=1THENP=0:RETURN
420 IFK=10THENPRINT:PRINT"pulsa una tecl
a";R$=INPUT$(1):GOTO140
430 LOCATE18,12:INPUTR$
440 RETURN
```

BREAKOUT

```
5 REM ********************
15 REM *     MSX-EXTRA      *
25 REM *     BREAKOUT       *
35 REM *       1985         *
45 REM *  (c)TOMAS SOLE     *
55 REM ********************
65 OPEN "GRP:" AS #1
75 V=5
85 LF=6
95 BR=0
105 SCR=0
115 SCREEN 2,0
125 COLOR 15,1,7
135 CLS
145 GOSUB 1065
155 GOSUB 855
165 GOSUB 955
175 BX=120
185 X=100
195 Y=80
205 HV=4
215 VV=4
225 REM BUCLE PRINCIPAL
235 T=STICK(0):IF T<>0 THEN GOSUB 405
```

```
245 HV=HV*2*(.5+(X<=18 OR X>=239))
255 IF Y<=8 THEN VV=-VV
265 IF 176=Y AND X+4>=BX AND X<=BX+12 TH
EN GOSUB 355
275 IF Y>184 THEN 685
285 Y=Y+VV
295 X=X+HV
305 PUT SPRITE 0,(X,Y),15,0
315 C=POINT (X+3,Y+3)
325 IF C>=2 THEN GOSUB 475
335 GOTO 225
345 REM BATE
355 VV=-VV
365 IF ABS(HV)=4 AND RND(1)<.5 THEN HV=S
GN(HV)*2
375 IF ABS(HV)=2 AND RND(1)<.5 THEN HV=S
GN(HV)*4
385 RETURN
395 REM MOVIMIENTO DE BATE
405 BX=BX+6*((T=7)-(T=3))
415 IF BX<=14 THEN BX=14
425 IF BX>=235 THEN BX=235
435 PUT SPRITE 1,(BX,181),11,1
445 PUT SPRITE 2,(BX+8,181),11,1
455 RETURN
465 REM DESTRUCCION DE LADRILLOS
475 VV=-VV
485 SCR=SCR+C
495 BR=BR+1
505 BEEP
515 TY=INT((Y-3)/8)*8+6
525 TX=INT ((X-12)/10)*10+14
535 LINE (TX,TY)-STEP (8,6),1,BF
545 DRAW "BM80,0"
555 LINE -STEP (48,8),11,BF
565 DRAW "BM80,0"
575 PRINT#1,SCR
585 IF BR=LF*22 THEN GOSUB 615
595 RETURN
605 REM MAS LADRILLOS
615 LF=LF+1
625 GOSUB 955
635 X=100:Y=80
```

```
645 HV=4:VV=4
655 PUT SPRITE 0,(X,Y),11,0
665 RETURN
675 REM PERDIDA DE UNA VIDA
685 DRAW "BM179,0"
695 LINE -STEP (66,8),11,BF
705 DRAW "BM179,0"
715 COLOR 1
725 V=V-1
735 PRINT#1,"VIDAS ";V
745 BX=120
755 PUT SPRITE 1,(BX,181),11,1
765 PUT SPRITE 2,(BX+8,181),11,1
775 IF V<>0 THEN 175
785 DRAW "BM100,100"
795 COLOR 13
805 PRINT#1,"FIN DEL JUEGO"
815 DRAW "BM100,110"
825 PRINT#1,"PULSE BARRA ESPACIO"
835 IF INKEY$=" " THEN RUN ELSE 835
845 REM FUERA DEL MURO
855 LINE (0,0)-STEP(18,191),11,BF
865 LINE (14,0)-STEP (236,8),11,BF
875 LINE (248,0)-(255,191),11,BF
885 DRAW "BM179,0"
895 COLOR 1
905 PRINT#1,"VIDAS";V
915 DRAW "BM29,0"
925 PRINT#1,"PUNTOS";SCR
935 RETURN
945 REM LADRILLOS
955 RESTORE 965
965 DATA 7,5,4,12,2,3,11,10,9,8,6,13,
 4,5,7,14,15
975 FOR A=1TO LF
985 READ C
995 FOR B=1 TO 22
1005 X=B*10+14
1015 Y=A*8+6
1025 LINE (X,Y)-STEP(8,6),C,BF
1035 NEXT B,A
1045 RETURN
1055 REM SPRITE BOLA
```

```
1065 RESTORE 1135
1075 FOR I=1 TO 8
1085 READ A$
1095 S$=S$+CHR$(VAL(A$))
1105 NEXT
1115 SPRITE$ (0)=S$
1125 R=RND(-TIME)
1135 DATA 0,0,24,60,60,24,0,0
1145 REM SPRITE BATE
1155 SPRITE$(1)=CHR$(255)+CHR$(255)+STRI
NG$(0,CHR$(0))
1165 RETURN
```

APOCALYPSE NOW

```
10 '*******APOCALYPSE  NOW ******
20 '*                            *
30 '* F.J. GUERRERO&J.C.GONZALEZ*
40 '*                            *
50 '*****************************
60 '&&&PANTALLA Y VARIABLES&&&
70 FC=4:SC=12:WQ=5:QQ=5:KEY OFF
80 OPEN"GRP:"AS#1
90 COLOR 15,7,1
100 SCREEN2,1:GOSUB950:COLOR 15,1,
1
110 PRESET(10,10):PRINT#1,"APOCALY
PSE NOW ":PRINT#1,"Este juego está
 pensado para 2 jugadores"
120 PRINT#1,"uno con teclado y otr
o con joystick.":PRINT#1,"El jugad
or del teclado mueve"
130 PRINT#1, "con las teclas de cu
rsor y dispara"
140 PRINT#1, "con la barra de espa
cios. "
150 PRESET(110,160):PRINT#1,"▲"
160 PRESET(110,172):PRINT#1,"█"
170 FOR Q=1TO 255:READDA:LINE(Q-1,
```

```
E)-(Q,DA),1:E=DA:NEXT Q:PRESET(60,
90):PRINT#1,"COMIENZA EL ATAQUE":P
AINT(10,10),1:CLOSE1
180 A2%=15:B2%=182:A1%=181:B1%=183
:FH=4:SH=4
190 ON SPRITE GOSUB 1500
200 '&&LECTURA DE JOYSTICK Y TECLA
DO&&
210 SPRITE ON:T2=3:GOSUB1800
220 T1=STICK(1):T2=STICK(0)
230 IF T1=1 THEN V1%=V1%+1
240 IF T1=2 THEN V1%=V1%+1:H1%=H1%
+1
250 IF T1=3 THEN H1%=H1%+1
260 IF T1=4 THEN V1%=V1%-1:H1%=H1%
+1
270 IF T1=5 THEN V1%=V1%-1
280 IF T1=6 THEN V1%=V1%-1:H1%=H1%
-1
290 IF T1=7 THEN H1%=H1%-1
300 IF T1=8 THEN V1%=V1%+1:H1%=H1%
-1
310 IF T2=1 THEN V2%=V2%+1
320 IF T2=2 THEN V2%=V2%+1:H2%=H2%
+1
330 IF T2=3 THEN H2%=H2%+1
340 IF T2=4 THEN V2%=V2%-1:H2%=H2%
+1
350 IF T2=5 THEN V2%=V2%-1
360 IF T2=6 THEN V2%=V2%-1:H2%=H2%
-1
370 IF T2=7 THEN H2%=H2%-1
380 IF T2=8 THEN V2%=V2%+1:H2%=H2%
-1
390 IF STRIG(1)=-1 AND FH>0 THEN B
1=1:F1=A1%:F2=B1%+9:G1=0:YY=1:FH=F
H-1:GOSUB 1760
400 IF STRIG(0)=-1 AND SH>0 THEN B
2=1:F3=A2%:F4=B2%+9:G2=0:XX=1:SH=S
H-1:GOSUB 1760
410 '&&&& DISPARO DE MISIL S&&&
420 IF FH=0 THEN FC=15
```

```
430 IF SH=0 THEN SC=15
440 A1%=A1%+(H1%/2):B1%=B1%+(V1%)
450 A2%=A2%+(H2%/2):B2%=B2%+(V2%)
460 IF B1=1 AND S1=0 THEN M1=-9:B1
=0
470 IF B2=1 AND S2=2 THEN M2=-9:B2
=0
480 IF B1=1 AND S1=1 THEN M1=+9:B1
=0
490 IF B2=1 AND S2=3 THEN M2=+9:B2
=0
500 IF T1=0 THEN GOSUB 1880
510 IF T2=0 THEN GOSUB 1900
520 IF F1+G1<0 OR F1+G1>255 THEN Y
Y=0:H1=0
530 IF F3+G2<0 OR F3+G2>255 THEN X
X=0:H2=0
540 G1=G1+M1:G2=G2+M2
550 IF YY=1 THEN PUT SPRITE5,(F1+G
1,F2),15:IF POINT(F1+G1,F2+3)=7 TH
EN YY=0:GOTO 570
560 IF YY=0 THEN PUT SPRITE5,(-20,
-20),15
570 IF XX=1 THEN PUT SPRITE6,(F3+G
2,F4),15:IF POINT(F3+G2,F4+3)=7 TH
EN XX=0:GOTO 600
580 '
590 IF XX=0 THEN PUT SPRITE6,(-12,
-13),15
600 IF H2%<0 THEN S2=2:PUT SPRITE3
,(-21,-22)
610 IF H1%<0 THEN S1=0:PUT SPRITE1
,(-30,-30)
620 IF H1%>0 THEN S1=1:PUT SPRITE0
,(-30,-30)
630 IF H2%>0 THEN S2=3:PUT SPRITE2
,(-21,-22)
640 PUT SPRITE S1,(A1%,B1%),FC:PUT
 SPRITE S2,(A2%,B2%),SC
650 D1=F2:D2=F4
660 '
670 IF F2<B2%+3 THEN F2=F2+3.5
```

```
680 IF F2>B2%+3 THEN F2=F2-3.5
690 IF F4<B1%+3 THEN F4=F4+3.5
700 IF F4>B1%+3 THEN F4=F4-3.5
710 '&&&&MUESTRA DEL HELICOPTERO&&
&
720 IF A1%<1 THEN A1%=246
730 IF A1%>246 THEN A1%=1
740 IF A2%<1  THEN A2%=246
750 IF A2%>246 THEN A2%=1
760 IF B1%<1 THEN B1%=1:V1%=0
770 IF V1%>183 THEN B1%=183:V1%=0
780 IF B2%<1 THEN B2%=1:V2%=0
790 IF B2%>183 THEN B2%=183:V2%=0
800 IF H1%>25 THEN H1%=25
810 IF H1%<-25 THEN H1%=-25
820 IF H2%>25 THEN H2%=25
830 IF H2%<-25 THEN H2%=-25
840 '
850 IF POINT(A1%,B1%)<>1 OR POINT(
A1%,B1%+8)<>1 OR POINT (A1%+8,B1%)
<>1 OR POINT(A1%+8,B1%+8)<>1 THEN
GOSUB 1920
860 IF POINT(A2%,B2%)<>1 OR POINT(
A2%,B2%+8)<>1 OR POINT(A2%+8,B2%)<
>1 OR POINT(A2%+8,B2%+8)<>1 THEN G
OSUB 1930
870 IF SO<>0 THEN SO=SO-1
880 IF    SO=0 THEN SPRITE ON
890 '
900 IF B1%>180 THEN FH=4:FC=4
910 IF B2%>180 THEN SH=4:SC=12
920 IF WQ=0 OR QQ=0 THEN GOSUB 196
0
930 GOSUB1800:GOTO 220
940 '&&&
950 A1$=CHR$(0)
960 A2$=CHR$(0)
970 A3$=CHR$(0)
980 A4$=CHR$(0)
990 A5$=CHR$(252)
1000 A6$=CHR$(33)
1010 A7$=CHR$(126)
```

```
1020 A8$=CHR$(112)
1030 A$=A1$+A2$+A3$+A4$+A5$+A6$+A6
$+A7$+A8$:SPRITE$(0)=A$
1040 A1$=CHR$(0)
1050 A2$=CHR$(0)
1060 A3$=CHR$(0)
1070 A4$=CHR$(0)
1080 A5$=CHR$(63)
1090 A6$=CHR$(132)
1100 A7$=CHR$(126)
1110 A8$=CHR$(14)
1120 A$=A1$+A2$+A3$+A4$+A5$+A6$+A7
$+A8$:SPRITE$(1)=A$
1130 A1$=CHR$(0)
1140 A2$=CHR$(0)
1150 A3$=CHR$(0)
1160 A4$=CHR$(252)
1170 A5$=CHR$(33)
1180 A6$=CHR$(126)
1190 A7$=CHR$(248)
1200 A8$=CHR$(72)
1210 A$=A1$+A2$+A3$+A4$+A5$+A6$+A7
$+A8$:SPRITE$(1)=A$
1220 A1$=CHR$(0)
1230 A2$=CHR$(0)
1240 A3$=CHR$(0)
1250 A4$=CHR$(63)
1260 A5$=CHR$(132)
1270 A6$=CHR$(126)

1280 A7$=CHR$(31)
1290 A8$=CHR$(18)
1300 A$=A1$+A2$+A3$+A4$+A5$+A6$+A7
$+A8$:SPRITE$(3)=A$
1310 A1$=CHR$(137)
1320 A2$=CHR$(74)
1330 A3$=CHR$(46)
1340 A4$=CHR$(18)
1350 A5$=CHR$(245)
1360 A6$=CHR$(20)
1370 A7$=CHR$(170)
1380 A8$=CHR$(68)
```

```
1390 A$=A1$+A2$+A3$+A4$+A5$+A6$+A7
$+A8$:SPRITE$(4)=A$
1400 A1$=CHR$(0)
1410 A2$=CHR$(0)
1420 A3$=CHR$(0)
1430 A4$=CHR$(32)
1440 A5$=CHR$(248)
1450 A6$=CHR$(32)
1460 A7$=CHR$(0)
1470 A8$=CHR$(0)
1480 A$=A1$+A2$+A3$+A4$+A5$+A6$+A7
$+A8$:SPRITE$(5)=A$:SPRITE$(6)=A$
1490 RETURN
1500 '
1510 '
1520 '
1530 '
1540 IFINT(A1%/10)=INT(A2%/10)AND
INT(B1%/10)=INT(B2%/10)THEN PUT SP

1550 IFINT((F1+G1)/10)=INT(A2%/10)
AND INT(F2/10)=INT(B2%/10)THEN PUT
SPRITES2,(-21,-22):PUT SPRITE5,(-2
0,-20):PUTSPRITE4,(A2%,B2%),15:GOS
UB1590:FOR T=0TO1000:NEXTT:PUT SPR
ITE4,(-1,-30):A2%=15:B2%=182:V2%=0
:H2%=0:WQ=WQ-1:RETURN
1560 IFINT((F3+G2)/10)=INT(A1%/10)
ANDINT(F4/10)=INT(B1%/10)THEN PUTS
PRITES1,(-30,30):PUTSPRITE6,(-20,4
0):PUTSPRITE4,(A1%,B1%),15:GOSUB15
90:FORT=0TO1000:NEXTT:PUT SPRITE4,
(-4,-20):A1%=181:B1%=183:V1%=0:H1%
=0:QQ=QQ-1:RETURN
1570 IFINT((F1+G1)/10)=INT((F3+G2)
/10)AND INT(F2/10)=INT(F4/10)THEN
PUT SPRITE5,(-20,-20):PUT SPRITE6,
(-20,40):PUT SPRITE4,(F1+G1,F2),15
:GOSUB1590:FORT=0TO1000:NEXTT:PUT
SPRITE4,(-4,-20):XX=0:YY=0:F4=-22:
F2=-12:RETURN
1580 SPRITE OFF:SO=1
1590 '&&&EXPLOSION&&&
```

```
1600 SOUND2,0:SOUND3,13
1610 SOUND4,255:SOUND5,15
1620 SOUND6,30:SOUND7,0
1630 SOUND8,16:SOUND9,16
1640 SOUND10,16:SOUND13,0
1650 FOR X=0TO 30:NEXT X
1660 SOUND12,56:SOUND13,0
1670 RETURN

1680 '&&&DATAS PARA LAS MONTAÑAS&&
&
1690 DATA120,200,11,120,143,140,14
0,130,120,100,100,200,200,200,200,
200,200,200,200,200,200,200,200,
200,200,200,200,200,200,200,200,20
0,200,200,200,200,190,150,120,120,
110,110,100,200,200
1700 DATA125,250,111,231,125,116,1
37,128,139,120,115,120,125,134,143
,153,153,182,131,180,255,137,138,1
89,197,173,138,127,128,137,157,167

1710 DATA100,200,100,100,200,100,9
0,120,150,190,180,170,170,165,174,
137,182,181,180,176,177,220,100,12
0,125,130,135,150,175,150,155,150,
155,170,175,177,178,179,180,181,18
2,183,184,184,183,182,182,181,180,
179,178,177,171,175,166,170,180,18
0,180,180,180
1720 DATA210,210,200,190,185,186,1
87,188,189,180,175,170,165,164,163
,163,163,162,161,160,155,157,158,1
59,147,133,128,187,188,187,187,187

1730 DATA100,100,110,120,143,140,1
40,130,120,100,100,200,200,200,200
,200,200,200,200,200,200,200,200,2
00,200,200,200,200,190,150,120,120
,110,110,100,200.200


1740 DATA200,200,200,200,200,200,1
90,190,190,190,190,200,190,185,184
```

```
,183,182,181,180,176,177,100,100,1
20,125,130,135,140,145,150,155,160
,165,170,175,177,178,179,180,181,1
82,183,184,184,183,182,182,181,180
,179,178,177,171,175,166,170,180,1
80,180,180,180
1750 DATA200,200,200,203,200,200,1
90,190,190,190,190,220,190,185,184
,183,182,181,185,178,137,103,150,1
20,195,130,135,140,145,150,155,160
,165,170,175,177,178,179,180,181,1
82,183,184,184,183,182,182,181,180
,179,178,197,171,175,196,180,190,1
80,190,170,180
1760 SOUND6,15:SOUND7,7:SOUND8,16:
SOUND9,16:SOUND10,16:SOUND11,0:SOU
ND11,0:SOUND12,16:SOUND13,0:RETURN

1770 '
1780 '
1790 '
1800 SOUND6,15
1810 SOUND7,135
1820 SOUND8,31
1830 SOUND9,31
1840 SOUND10,31
1850 SOUND11,0:SOUND12,1
1860 SOUND13,12
1870 RETURN


1880 IF H1%<0 THEN H1%=H1%+.5ELSEI
F H1%>0THEN H1%=H1%-.5
1890 V1%=V1%+.8:RETURN
1900 IF H2%<0 THENH2%=H2%+.5ELSE I
F H2%>0 THEN H2%=H2%-.8
1910 V2%=V2%+.5:RETURN
1920 PUT SPRITES1,(-20,0):PUT SPRI
TE4,(A1%,B1%),15:GOSUB1590:FOR T=0
TO1000:NEXT T:A1%=181:B1%=183:PUT
SPRITE4,(-20,-0):V1%=0:H1%=0:WQ=WQ
-1:RETURN
1930 PUT SPRITE S2,(-10,-4):PUT SP
```

```
RITE4,(A2%,B2%),15:GOSUB1590:FOR T
=0TO 1000:NEXT T:A2%=15:B2%=182:PU
T SPRITE4,(-10,-4):V2%=0:H2%=0:QQ=
QQ-1:RETURN
1940 '
1950 '
1960 BEEP:CLS:PLAY"","","o2v15ACEG
+2ECA"
1970 SCREEN0
1980 IFQQ=WQ THENPRINT"EMPATE"
1990 IF QQ>WQ THENPRINT"GANADOR EL
 JUGADOR DEL TECLADO"
2000 IF QQ<WQ THENPRINT"GANADOR EL
 JUGADOR DEL JOYSTICK"
2010 LOCATE10,7:PRINT"OTRA PARTIDA
(S/N)"
2020 INPUT T$
2030 IF T$="S"THEN RUN
2040 IFT$<>"S" THEN 2050:PRINT"HAS

2040 IFT$<>"S" THEN 2050:PRINT"HAS
TA OTRA CAMPEONES"
2050 END
```

EL ROBOT SALTARIN

```
10  ' *********************
20 ' * EL ROBOT SALTARIN  *
30 ' * POR F.JOSE SALAZAR *
40 ' * INSTRUCCIONES *    *
50 ' *********************
60 CLS:SCREEN 0
70 PRINT:PRINT"    EL ROBOT SALTARIN

80 PRINT:PRINT
90 PRINT"El objetivo del juego es conseg
uir el mayor numero de flechas posibles.
  Para ello debes coger las llaves que
se encuentran en el 1 piso y 2 piso y ll
egar hasta la puerta."
100 PRINT:PRINT"Tienes 5 robots;lograras
 uno extra   cada vez que consigas algun
a flecha,y 10000 puntos y 3 robots mas c
ada vez que pases la ultima flecha."
110 PRINT"Al principio posees 5 robots,
cuando te quede uno, en el angulo superi
or  derecho aparecera un 1."
120 PRINT:PRINT:PRINT"   PULSA --A-- PAR
A CONTINUAR"
130 IF INKEY$="A" OR INKEY$="a" THEN 140
 ELSE 130
140 PRINT:PRINT"Debes tener cuidado porq
```

```
ue hay boños invisibles que cuando consi
gues una  flecha se suman a tu puntuacio
n, pero si llegan a 0,acabaras la partid
a.";
150 PRINT"Al principio cuentas con 3000
bonos,   cuando lleguen a 1000, en el an
gulo   superior derecho se visualizara u
na campana naranja, y cuando lleguen a
500 una roja, que te avisara del     pel
igro."
160 PRINT:PRINT"Cada vez que consigas un
a flecha, los bonos se repondran y el ni
vel de    dificultad aumentara.Puedes sa
ltar    hacia adelante y hacia detras.
     BUENA SUERTE"
170 PRINT:PRINT:PRINT"    PULSA --A-- PAR
A CONTINUAR"
180 IF INKEY$="A" OR INKEY$="a" THEN 190
 ELSE 180
190 PLAY "SFCF"
200 MN=1:GOTO 210
210 J=170:B=0:L=30:P=162:K=6:HH=40:F=162
:II=5:IO=5:YO=3:PP=162:PT=135:Q=148:W=16
2:FL=3:PE=55:TQ=5:S=0:FH=0:CE=3000:RT=10
:LO=7:LU=7
220 ON SPRITE GOSUB 1790
230 COLOR 10,1,1
240 SCREEN 2,2
250 '
260 ' * DATAS DE SPRITES *
270 '
280 DATA 0,0,0,0,0,0,0,0
290 DATA 1,3,1,0,0,0,0,0
300 DATA 0,0,0,0,0,0,0,204
310 DATA 254,183,254,252,252,48,72,48
320 DATA 3,7,2,3,3,1,3,5
330 DATA 11,19,1,3,6,12,28,0
340 DATA 192,224,64,192,192,128,192,160
350 DATA 208,200,128,192,96,48,56,0
360 DATA 0,0,3,0,0,0,0,0
370 DATA 0,0,0,0,0,0,0,0
380 DATA 0,135,254,135,0,0,0,0
390 DATA 0,0,0,0,0,0,0,0
```

```
400 DATA 1,59,46,56,0,0,0,0
410 DATA 0,0,0,0,0,0,0,0
420 DATA 4,142,219,112,32,0,0,0
430 DATA 0,0,0,0,0,0,0,0
440 DATA 0,0,0,0,0,0,0,0
450 DATA 0,0,0,0,0,0,0,0
460 DATA 0,0,0,0,0,0,31,17
470 DATA 31,4,4,4,4,28,12,28
480 DATA 0,1,3,7,7,7,7,7
490 DATA 5,5,7,7,7,7,7,7
500 DATA 232,244,250,253,253,253,253,253

510 DATA 253,253,253,253,253,253,253,253

520 DATA 0,0,0,1,0,0,0,1
530 DATA 1,0,0,0,0,0,0,0
540 DATA 3,29,126,254,254,124,124,152
550 DATA 136,0,0,0,0,0,0,0
560 DATA 0,0,0,0,0,0,0,0
570 DATA 0,0,0,0,0,0,0,0
580 DATA 7,7,3,3,3,3,3,3
590 DATA 0,0,0,0,0,0,0,0
600 FOR Y=1 TO 8
610 A$="":FOR X=1 TO 32:READ D:A$=A$+CHR
$(D):NEXT X:SPRITE$(Y)=A$
620 NEXT Y
630 LINE (0,0)-(19,195),1,BF
640 LINE (235,0)-(255,192),1,BF
650 LINE (50,0)-(200,10),7,B
660 FOR JK=50 TO 200 STEP 30
670 LINE (JK,0)-(JK,10),7
680 NEXT JK
690 '
700 ' * CREACION DEL SUELO *
710 '
720 OPEN "GRP:" AS#1
730 FOR F=20 TO 235 STEP 8
740 FOR D=15 TO 190 STEP 55
750 PSET (F,D):PRINT#1,CHR$(215)
760 NEXT D,F:GOTO 820
770 K=-K+2:GOTO 820
780 L=30:P=P
790 '
```

```
800 ' * VISUALIZACION DE SPRITES *
810 '
820 PUT SPRITE 0,(180,162),3,1
830 PUT SPRITE 7,(60,162),3,1
840 PUT SPRITE 8,(140,136),7,4
850 PUT SPRITE 1,(L,P),11,2
860 PUT SPRITE 2,(100,136),7,4
870 PUT SPRITE 11,(215,85),9,II
880 PUT SPRITE 4,(215,140),9,IO
890 PUT SPRITE 12,(180,107),3,1
900 PUT SPRITE 13,(125,107),3,1
910 PUT SPRITE 14,(55,107),2,1
920 PUT SPRITE 5,(215,30),15,6
930 PUT SPRITE 15,(75,80),7,4
940 PUT SPRITE 16,(160,80),7,4
950 PUT SPRITE 6,(HH,F),FL,3
960 PUT SPRITE 17,(60,52),3,1
970 PUT SPRITE 18,(90,52),3,1
980 PUT SPRITE 19,(140,52),2,1
990 PUT SPRITE 20,(40,24),7,4
1000 PUT SPRITE 21,(110,24),7,4
1010 PUT SPRITE 22,(170,24),7,4
1020 GOTO 1070
1030 W=162:Q=148:P=P+110:LO=7:LU=7
1040 '
1050 ' * MOVIMIENTO DEL ROBOT *
1060 '
1070 D=STICK(0):SPRITE ON
1080 HH=HH-YO:IF HH<5 THEN GOTO 1260
1090 IF D=3 THEN BEEP:L=L+2:P=P+2
1100 IF D=7 THEN BEEP:L=L-2:P=P+2
1110 IF D=1 THEN BEEP:L=L+5:P=P-K:L=L-5:
IF P<PT THEN K=-K+2
1120 IF D=2 THEN BEEP:L=L+3:P=P-K:IF P<P
T THEN K=-K+2
1130 IF D=8 THEN BEEP:L=L-3:P=P-K:IF P<P
T THEN K=-K+2
1140 IF D=0 THEN PUT SPRITE 1,(L,P),15,2
:P=P+2
1150 IF D=5 THEN P=P+2
1160 IF D=4 THEN P=P+2
1170 IF D=6 THEN P=P+2
1180 IF L<20 THEN L=20
```

```
1190 IF P<20 THEN P=20
1200 IF L>200 AND P>70 AND P<130 THEN GO
TO 1530
1210 IF L>200 AND P>140 THEN GOTO 1540
1220 IF L>200 AND P<70 AND P>10 THEN 167
0
1230 IF P>PP THEN P=PP:GOTO 770
1240 GOTO 820
1250 BEEP:GOTO 770
1260 PLAY"SB":HH=200:A=INT(RND(1)*2)+1
1270 B=INT(RND(1)*7)
1280 CE=CE-100:IF CE<=0 THEN 1950
1290 IF CE<=1000 THEN PUT SPRITE 24,(15,
0),9,LO
     1300 IF CE<=500 THEN PUT SPRITE 25,
(30,0),8,LO
1310 '
1320 ' * VELOCIDAD,COLOR Y ALTURA *
1330 '        * DE LAS FLECHAS *
1340 GG=INT(RND(1)*5)
1350 IF GG=0 THEN FL=3
1360 IF GG=1 THEN FL=7
1370 IF GG=2 THEN FL=9
1380 IF GG=3 THEN FL=10
1390 IF GG=4 THEN FL=11
1400 IF GG=5 THEN FL=15
1410 IF B=0 THEN YO=6+MN
1420 IF B=1 OR B=2 THEN YO=2+MN
1430 IF B=2 OR B=3 THEN YO=3+MN
1440 IF B=4 OR B=5 THEN YO=4+MN
1450 IF B=6 THEN YO=5+MN
1460 IF B=7 THEN YO=7+MN
1470 IF A=1 THEN F=Q
1480 IF A=2 THEN F=W
1490 GOTO 1090
1500 '
1510 ' * LLAVE COGIDA *
1520 '
1530 II=10:S=S+100
1540 SOUND 0,0:SOUND 1,5:SOUND 2,0
1550 SOUND 3,13:SOUND 4,255:SOUND 5,15
1560 SOUND 6,30:SOUND 7,0
1570 SOUND 8,16:SOUND 9,16:SOUND 10,16
```

```
1580 SOUND 11,0:SOUND 12,5:SOUND 13,0
1590 FOR T=1 TO 20:NEXT T
1600 SOUND 12,56:SOUND 13,0
1610 S=S+100:IO=10:L=20:P=P-55:PP=PP-55:
PT=PT-55
1620 Q=Q-55:W=W-55
1630 GOTO 770
1640 '
1650 ' * CONSECUCION DE LA FLECHA *
1660 '
1670 LO=10:PUT SPRITE 24,(15,0),9,LO:LU=
10:PUT SPRITE 25,(30,0),8,LU:S=S+CE:CE=3
000:FH=FH+1:S=S+300:IF PE>=205 THEN 2090

1680 PUT SPRITE 27,(PE,3),5,3
1690 PE=PE+30
1700 PUT SPRITE 26,(100,80),7,4
1710 MN=MN+1:IF MN>=7 THEN MN=7
1720 TQ=TQ+1:RT=10:PUT SPRITE 29,(210,0)
,15,RT
1730 L=30:PP=162:PT=135
1740 II=5:IO=5
1750 GOTO 1030
1760 '
1770 ' * SOLAPAMIENTO DE FIGURAS *
1780 '
1790 SPRITE OFF
1800 SOUND 0,0:SOUND 1,5
1810 SOUND 2,0:SOUND 3,13
1820 SOUND 4,255:SOUND 5,15
1830 SOUND 6,30:SOUND 7,0
1840 SOUND 8,16:SOUND 9,16:SOUND 10,16
1850 SOUND 11,0:SOUND 12,5:SOUND 13,0
1860 FOR GH=1 TO 45:NEXT GH:SOUND 12,56:
SOUND 13,0
1870 FOR E=1 TO 150:NEXT E
1880 HH=200:TQ=TQ-1:IF TQ=0 THEN 1950
1890 IF TQ=1 THEN RT=8
1900 PUT SPRITE 29,(210,0),15,RT
1910 RETURN 780
1920 '
1930 ' * FINAL PARTIDA *
1940 '
```

```
1950 CLS:SCREEN 0
1960 LOCATE 11,3:COLOR 9:PRINT"SCORE"
1970 LOCATE 11,5:COLOR 9:PRINT;S;
1980 LOCATE 7,7:COLOR 9:PRINT "HAS LOGRA
DO";FH;:IF FH=1 THEN PRINT"FLECHA" ELSE
PRINT"FLECHAS"
1990 IF S<4000 THEN LOCATE 11,9:PRINT"!Q
UE MAL!"
2000 IF S>4000 AND S<8000 THEN LOCATE 11
,9:PRINT "!BIEN!"
2010 IF S>8000 AND S<17000 THEN LOCATE 1
1,9:PRINT "!MUY BIEN!"
2020 IF S>17000 THEN LOCATE 11,9:PRINT "
!YA TE VALE!"
2030 LOCATE 5,14:COLOR 15:PRINT"PULSA LA
 BARRA PARA EMPEZAR"
2040 FOR J=1 TO 400:NEXT J
2050 IF INKEY$=" " THEN RUN ELSE 2050
2060 '
2070 ' * PASE DE FLECHA *
2080 '
2090 PLAY"ABDBDBA":PE=55:S=S+10000:TQ=TQ
+2:RT=10:PUT SPRITE 29,(210,0),15,RT
2100 GOTO 1680
```

EL ARCHIVO EN CASA

```
10 SCREEN 3 :COLOR 15,4,11
15 LINE (10,10)-(250,182),9,B
17 LINE (15,15)-(245,177),9,B
19 PAINT (11,11),9,9
20 OPEN "GRP:" AS #1
30 PRESET (25,80)
40 PRINT #1, "ARCHIVO"
50 PLAY "ABCDEFFEDC"
55 CLOSE
60 FOR V=1 TO 4000 :NEXT
70 GOTO 155
100 REM **********************
110 REM * EL ARCHIVO EN CASA *
120 REM * SUPER JUEGOS       *
130 REM * EXTRA              *
140 REM **********************
150 REM
155 CLS :SCREEN 0 :COLOR 1,7
160 CLEAR 3000
170 REM DIMENSIONA  SERIES PARA CARGAR
 REGISTROS Y NOMBRES DE REGISTRO
180 DIM R$(500),N$(500)
190 COLOR 1,7
200 CLS
210 REM IMPRIME  MENU
220 LOCATE 9,1
```

```
230 PRINT "MENU DEL ARCHIVO"
240 LOCATE 0,4
250 PRINT "1.  Introducir nuevos datos o
 iniciacion de un archivo": PRINT
260 PRINT "2. Corregir datos":PRINT
270 PRINT "3. Visualizacion  de datos en
 la pantalla":PRINT
280 PRINT "4. Listado de datos en la imp
resora":PRINT
290 PRINT "5. Busqueda de datos":PRINT
300 PRINT "6. Clasificar datos":PRINT
310 PRINT "7. Cargar datos de la cinta":
PRINT
320 PRINT "8. Grabar en la cinta":PRINT
330 LOCATE 0,23
340 PRINT "Seleccione la opcion numero..
."
350 REM VALIDA LA RESPUESTA
360 A$=INKEY$
370 IF A$="" THEN 360
380 BEEP
390 IF A$<"1" OR A$>"8" THEN 370
400 ON VAL(A$) GOTO 420,610,1210,1330,14
40,1860,1980,2120
410 REM NUEVO REGISTRO
420 CLS
430 LOCATE 0,6
440 INPUT "Es esta una nueva ficha";Q$
450 IF LEFT$(Q$,1)="s" OR LEFT$(Q$,1)="S
" THEN J=1 ELSE J=J+1
460 CLS
470 LOCATE 0,3
480 PRINT " Introducir todos los datos d
e la    ficha-indique FIN para finalizar
."
490 PRINT:PRINT
500 LINE INPUT "Nombre de la ficha? ";N$
(J)
510 IF N$(J)="FIN" OR N$(J)="fin" THEN 5
80
520 PRINT
530 LINE INPUT "Informacion? ";R$(J)
540 IF R$(J)="FIN" OR R$(J)="fin" THEN 5
```

```
80
550 PRINT:PRINT
560 J=J+1
570 GOTO 500
580 J=J-1
590 GOTO 200
600 REM MODIFICACION DE REGISTROS
610 CLS
620 N=0
630 LOCATE 0,3
640 INPUT "Nombre de la ficha";T$
650 PRINT:PRINT
660 M=1
670 IF T$=N$(M) THEN N=M:GOTO 740
680 M=M+1
690 IF M<>J THEN 670
700 IF N<>0 THEN 720
710 PRINT "Esta ficha  ";T$;" no ha sido
 cargada"
720 GOSUB 2290
730 GOTO 200
740 PRINT N$(N):PRINT
750 PRINT R$(N):PRINT
760 INPUT "Esta ficha";Q$
770 IF LEFT$(Q$,1)="S" OR LEFT$(Q$,1)="s
" THEN 800
780 GOTO 680
790 REM MODIFICAR MENU
800 CLS
810 LOCATE 0,4
820 PRINT "1. Borrar datos":PRINT
830 PRINT "2. Corregir nombre de la fich
a" :PRINT
840 PRINT "3. Corregir informacion" :PRI
NT
850 PRINT "4. Añadir informacion":PRINT
860 LOCATE 0,14
870 PRINT "Seleccionar la opcion...":PRI
NT
880 A$=INKEY$
890 IF A$="" THEN 880
900 BEEP
910 IF A$<"1" OR A$>"4" THEN 880
```

```
920 ON VAL (A$) GOTO 940,1050,1100,1150
930 REM BORRAR REGISTROS
940 FOR M=N TO J
950 N$(M)=N$(M+1)
960 R$(M)=R$(M+1)
970 NEXT M
980 N$(J)=""
990 R$(J)=""
1000 J=J-1
1010 PRINT "Grabacion borrada"
1020 GOSUB 2290
1030 GOTO 200
1040 REM MODIFICAR NOMBRE
1050 INPUT "Ficha nueva";N$(N)
1060 PRINT:PRINT"Nombre cambiado"
1070 GOSUB 2290
1080 GOTO 200
1090 REM MODIFICAR INFORMACION
1100 INPUT "Nueva informacion";R$(N)
1110 PRINT: PRINT"Información cambiada"
1120 GOSUB 2290
1130 GOTO 200
1140 REM AÑADIR INFORMACION
1150 INPUT "Nueva informacion";T$
1160 R$(N)=R$(N)+" - "+T$
1170 PRINT : PRINT "Nueva informacion añ
adida"
1180 GOSUB 2290
1190 GOTO 200
1200 REM DISPLAYA REGISTROS
1210 CLS
1220 D$=STRING$(36,219)
1230 FOR M=1 TO J
1240 PRINT:PRINT"Ficha cargada:"N$(M)
1250 PRINT:PRINT R$(M)
1260 PRINT:PRINT D$
1270 FOR DE=1 TO 100:NEXT DE
1280 GOSUB 2320
1290 NEXT M
1300 GOSUB 2290
1310 GOTO 200
1320 REM LISTADO DE REGISTROS POR IMPRES
ORA
```

```
1330 CLS
1340 LOCATE 0,6
1350 LINE INPUT "Entrar con la impresora
 preparada";NU$
1360 FOR M=1 TO J
1370 LPRINT
1380 LPRINT "Nombre de la ficha:";N$(M)
1390 LPRINT
1400 LPRINT R$(M)
1410 LPRINT
1420 NEXT M
1430 GOTO 200
1440 REM BUSCA REGISTROS
1450 CLS
1460 LOCATE 0,6
1470 PRINT "1. Nombre de la ficha":PRINT

1480 PRINT "2. Menu" : PRINT
1490 LOCATE 0,12
1500 PRINT "Seleccionar opcion...":PRINT

1510 A$=INKEY$
1520 IF A$="" THEN 1510
1530 BEEP
1540 IF A$<"1" OR A$>"2" THEN 1510
1550 ON VAL(A$) GOTO 1560,1700
1560 INPUT "Nombre de la ficha";T$
1570 PRINT
1580 F=0
1590 FOR M=1 TO J
1600 IF N$(M)<>T$ THEN 1640
1610 PRINT "Nombre de la ficha:";N$(M)
1620 PRINT:PRINT R$(M)
1630 F=1
1640 NEXT M
1650 IF F=1 THEN 1670
1660 PRINT:PRINT"Grabacion no cargada"K
1670 GOSUB 2290
1680 GOTO 200
1690 REM CADENAS
1700 INPUT "Busqueda de un dato";T$
1710 F=0
1720 FOR M=1 TO J
```

```
1730 X=INSTR(N$(M),T$)
1740 IF X<>0 THEN 1770
1750 X=INSTR(R$(M),T$)
1760 IF X=0 THEN 1800
1770 PRINT:PRINT"Nombre de la ficha:";N$
(M)
1780 PRINT:PRINTR$(M):PRINT
1790 F=1
1800 NEXT M
1810 IF F=1 THEN 1830
1820 PRINT:PRINT"Grabacion no cargada"
1830 GOSUB 2290
1840 GOTO 200
1850 REM CLASIFICA REGISTROS,OBSERVESE E
L USO DE LA SENTENCIA SWAP QUE INTERCAMB
IA EL VALOR DE DOS VARIABLES,EN ESTE CAS
O ENTRE UNA SERIE DE SUBINDICE M,Y OTRA
DE SUBINDICE M+1
1860 CLS
1870 LOCATE 0,6
1880 PRINT "Sorting..."
1890 M=1
1900 IF N$(M)<N$(M+1) THEN 1940
1910 SWAP N$(M),N$(M+1)
1920 SWAP R$(M),R$(M+1)
1930 GOTO 1890
1940 M=M+1
1950 IF M<>J THEN 1900
1960 GOTO 200
1970 REM CARGA DESDE EL CASSETTE
1980 CLS
1990 LOCATE 0,6
2000 PRINT "Cargando desde el cassette"
:PRINT
2010 LINE INPUT "Presione  PLAY  y  RETU
RN  para cargar desde el cassette";NU$
2020 PRINT:PRINT "Cargando..."
2030 OPEN "CAS:FILE" FOR INPUT AS 1
2040 INPUT #1,J
2050 FOR M=1 TO J
2060 INPUT#1,N$(M)
2070 INPUT #1,R$(M)
2080 NEXT M
```

```
2090 CLOSE
2100 GOTO 200
2110 REM GRABA EN CASSETTE
2120 CLS
2130 LOCATE 0,6
2140 PRINT "Grabar en el cassette":PRINT

2150 LINE INPUT "Presione  REC+PLAY  y
RETURN  para grabar en el cassette";NU$
2160 PRINT :PRINT"Grabando..."
2170 MOTOR ON
2180 GOSUB 2290
2190 MOTOR OFF
2200 OPEN "CAS:FILE" FOR OUTPUT AS 1
2210 PRINT#1,J
2220 FOR M=1 TO J
2230 PRINT#1,N$(M)
2240 PRINT#1,R$(M)
2250 NEXT M
2260 CLOSE
2270 GOTO 200
2290 FOR DE=1 TO 2500:NEXT DE
2296 REM RETARDO PARA SALIDA A SUBRUTINA

2290 FOR DE=1 TO 2500:NEXT DE
2300 RETURN
2310 REM APRETAR TECLA PARA SUBRUTINA
2320 Z$=INKEY$
2330 IF Z$="" THEN 2360
2340 Z$=INKEY$
2350 IF Z$="" THEN 2340
2360 RETURN
```

INDICE

Pág.

PROGRAMAS

Pág.

LOS SECRETOS DEL MSX
se acabó de imprimir en los
talleres de Megrasa en
Barcelona la
segunda quincena de Julio de
MCMLXXXV